TAMPEREEN TEKNILLINEN YLIOPISTO

OSSI TAATILA

GATEWAY ADAPTERS FOR INTEGRATING WIRELESS SENSOR NETWORKS

Master of Science Thesis

Examiners: Prof. Marko Hännikäinen,
Dr. Tech. Jukka Suhonen
Subject approved by Department
Council 9.5.2012

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Automation Technology
OSSI TAATILA: Gateway Adapters for Integrating Wireless Sensor Networks
Master of Science Thesis, 52 pages
September 2012
Major: Programmable Platforms and Devices
Examiners: Prof. Marko Hännikäinen, Dr. Tech. Jukka Suhonen
Keywords: Wireless Sensor Network, gateway, adapter, technology integration

Wireless sensor networks have grown more and more popular in the past few years. There are sensor networks available from several manufacturers. A wireless sensor network is usually tailored for a certain purpose and because of it's sensors, application specific radio range, data transfer speed, or other features it is unsuitable for other applications. In applications where several physical quantities are measured, it is hard to find one sensor network technology which would cover all of the requirements. Co-operative use of one or more different wireless sensor network technology is required in these applications.

This thesis reports the design and prototype implementation of an adaptation layer for connecting several wireless sensor network technologies to the same gateway. An adaptation layer consists of technology specific adapters, which convert the technology specific measurement messages to one unified message format understood by the gateway. This makes it possible to use the measurement data similarly regardless of the sensor network it is from.

This thesis presents adapter implementations for ZigBee, Z-Wave, and Bluetooth networks. The adapters are tested as a part of a larger ensemble, where the measurement data is used in cloud services. In addition this thesis introduces requirements for designing and implementing new sensor network adapters.

The results of the thesis show that several different wireless sensor network technologies can be used co-operatively with an adaptation layer. The adaptation layer also separates the sensor network integrations to the gateway from the rest of the development, speeding up the development of the gateway and end-user services. In addition the adaptation layer makes applications independent of one sensor network manufacturer, and makes it easy to add and change sensor networks to already deployed applications.

On the downside, the adaptation layer increases delays between a wireless sensor network and a gateway. It also adds a layer of complication to the architecture, creating one additional unreliability factor.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Automaatiotekniikan koulutusohjelma
OSSI TAATILA: Yhdyskäytävän sovittimet langattomien sensoriverkkojen integrointiin
Diplomityö, 52 sivua
Syyskuu 2012
Pääaine: Ohjelmoitavat alustat ja laitteet
Tarkastajat: Prof. Marko Hännikäinen, TkT Jukka Suhonen
Avainsanat: Langaton sensoriverkko, yhdyskäytävä, sovitin, teknologioiden integrointi

Langattomat sensoriverkot ovat yleistyneet viime vuosina. Tarjolla on sensoriverkkoja useilta eri valmistajilta. Yksittäinen sensoriverkko on yleensä suunnattu tiettyyn tarkoitukseen, eikä antureidensa, tai käyttötapaukseen mukautetun kantomatkan, tiedonsiirron nopeuden, tai muiden ominaisuuksien takia sovellu muihin käyttötapauksiin. Sovelluksiin joissa mitataan useita fyysisiä suureita on vaikea löytää yhtä sensoriverkkoa joka kattaisi kaikki vaatimukset. Tällaisiin sovelluksiin vaaditaan yhden tai useamman eri sensoriverkkoteknologian yhteiskäyttöä.

Tämä diplomityö esittelee sovitinkerroksen sunnittelun ja prototyyppitoteutuksen usean sensoriverkkoteknologian liittämiseen yhteen yhdyskäytävään. Sovitinkerros koostuu teknologiakohtaisista sovittimista, jotka muuntavat sensoriverkkojen omat mittausviestimuodot yhteiseen yhdyskäytävän ymmärtämään muotoon. Tämä mahdollistaa mittaustiedon käytön jatkosovelluksissa samalla tavalla riippumatta siitä, mistä sensoriverkosta se on peräisin.

Tässä diplomityössä esitellään sovittimien toteutus ZigBee, Z-Wave ja Bluetooth verkoille. Sovittimet testataan osana suurempaa kokonaisuutta, jossa sensoriverkkojen tuottamaa tietoa käytetään pilvipalveluissa. Lisäksi työssä esitetään yleiset vaatimukset uusien sensoriverkkosovittimien suunnitteluun ja toteutukseen.

Työn tulokset osoittavat, että useita eri teknologian sensoriverkkoja voidaan käyttää yhdessä sovitinkerroksen avulla. Lisäksi sovitinkerros eriyttää sensoriverkkojen integroinnin yhdyskäytävään muusta kehitystyöstä nopeuttaen huomattavasti sekä yhdyskäytävän, että loppusovellusten kehittämistä. Sovitinkerros mahdollistaa myös riippumattomuuden yhden valmistajan tuotteista, sekä sensoriverkkojen vaihtamisen ja lisäämisen helposti jo käytössä oleviin sovelluksiin.

Sovitinkerroksen varjopuolena on sen aiheuttama viiveen kasvu sensoriverkon ja

yhdyskäytävän välillä. Se myös luo yhden epävarmuustekijän lisää arkkitehtuuriin lisäämällä sen monimutkaisuutta.

PREFACE

This thesis was carried out at Tampere University of Technology in the Department of Computer Systems in 2012 in the TUTWSN research group.

I would like to express my gratitude to my thesis supervisors Prof. Marko Hännikäinen and Dr. Tech. Jukka Suhonen, and to my colleague M.Sc. Teemu Laukkarinen for their guidance and support during the thesis work. I am also grateful to M.Sc. Olli Kivelä and M.Sc. Ilkka Kautto for mentoring me during my first months at the Department. I would also like to thank all of my colleagues for the friendly atmosphere.

I would like to thank my parents Merja and Seppo for both financial and mental support during my studies. Also, I would like to thank the rest of my family and friends for their patience and endless support.

Tampere, September 7th, 2012

Ossi Taatila

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

LIST OF ABBREVIATIONS

ACF	Authentication and Capability Format
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BLE	Bluetooth Low Energy
CSV	Comma-Separated Values
GSN	Global Sensor Network
GWG	Generic WSN Gateway
IM	Instant Messaging
MAC	Medium Access Control
NASC	Node Actuator and Sensor Control
REST	REpresentational State Transfer
SIDF	Sensor Information Data Format
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TUT	Tampere University of Technology
UI	User Interface
USB	Universal Serial Bus
WAS	Distributed interoperable wireless surveillance, monitoring, and alarm system
WLAN	Wireless Local Area Network
WSN	Wireless Sensor Network
XML	eXtensible Markup Language

1. INTRODUCTION

The research and development of Wireless Sensor Networks (WSN) has yielded a vast application space and numerous deployments have been tested in academic and commercial world. Applications have been tested in areas such as agriculture [1], health care [2], military [3] and home surveillance [4]. The grown application space and increased commercial potential has lead to mass market. This has increased the number of manufacturers, standards and different technologies available. The use of several competing standards and immature technologies has lead to products incompatible with each other.

A WSN consists of independent devices, called *nodes*, that are equipped with sensors which measure physical quantities, such as temperature or humidity, and possible *actuators* that control other devices [5]. This thesis concentrates on resource constrained low-power WSNs that have small form factor nodes with embedded software [6]. Battery life for these nodes is usually measured in years and months rather than hours or days. The nodes communicate through a wireless radio link and automatically form an ad hoc network, where measurement data is transferred and routed hop by hop towards data collection points. Nodes communicate with each other at ranges from few meters up to kilometers [7]. A network can consist from just few nodes to thousands of nodes. The small form factor, long battery life and autonomous networking makes WSNs easy to deploy and maintain.

1.1 Typical WSN architecture

A widely used WSN architecture model is presented in Figure 1.1. This architecture is also used in this thesis. The WSN architecture consists of the network itself and the server infrastructure. The network consists of one or more nodes. A *sink node* is a special node at the edge of the network which connects the network to the outside world [6]. In a *multihop* WSN, nodes route data through other nodes, until the data arrives at the sink node.

The server infrastructure is used to store and process the measured data [6]. It

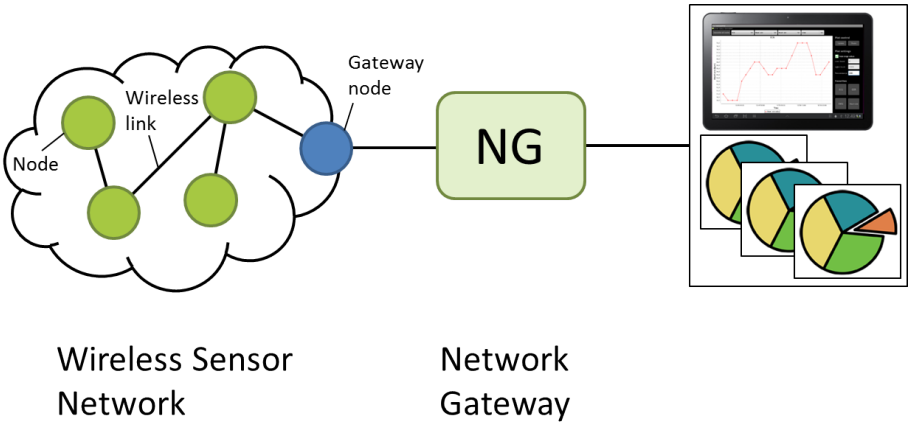

Figure 1.1: A typical end-to-end WSN architecture starting from a node and ending to an end-user service.

consists of a network gateway which can have a database for storing all measurements. It communicates with the sink node and receives all measurements from the network. The gateway is also connected to Internet or a local network to access the measurement data from the outside world. The gateway provides the data to end-user services in real-time or from the database based on queries. A gateway may support the subscription of alarms from sensor values, for example when a value exceed specified boundaries.

End-user services use the measurement data from the nodes. They can vary from a simple desktop software that graphically interprets the measurement data to servers that refine the data use it for calculation, for example weather forecasting. Refining the measurement data to reasonable and useful information is the main purpose of end-user services.

The grown application space of WSNs has made it impossible for one WSN technology to be a fixed solution that could be used in all applications [8]. The requirements of the WSN greatly vary between applications. For example nodes used for biomed-

Figure 1.2: Different types of WSN applications: weather station for outdoor conditions (left), cattle breeding monitoring [9] (middle) and human heart monitoring [10] (right)

ical measurements in short sports training sessions have different requirements on radio range and battery life compared to nodes that measure soil moisture on a farm for years. This has lead to many different WSN technologies, as illustrated in Figure 1.2, that are not compatible with each other. Some technologies are meant for a very specific application as some might be designed to be all-around solutions.

1.2 Multi-technology WSN architectures

Combining several WSN technologies enable applications with diverse requirements that one WSN technology cannot provide [11]. An application may need to measure different physical quantities and the distances between nodes can vary. For example in a home automation application nodes could monitor indoor temperature, humidity, energy consumption and control curtains and power switches. In the same application there could be nodes outdoors with longer distances monitoring motion, outside temperature and car parking. There might not be a WSN technology that has nodes or sensors able to cover all of the requirements. In these cases two or more different WSN technologies, incompatible with each other, have to be used together. The requirements for the application can also change after the initial deployment and new WSN technologies need to be integrated into the application.

Multiple WSN technologies can be integrated into one gateway with an adaptation layer between the gateway and the WSNs as illustrated in Figure 1.3. This method is used in this thesis. The adaptation layer consists of adapters which convert the WSN technology specific measurement data to an unified format understood by the

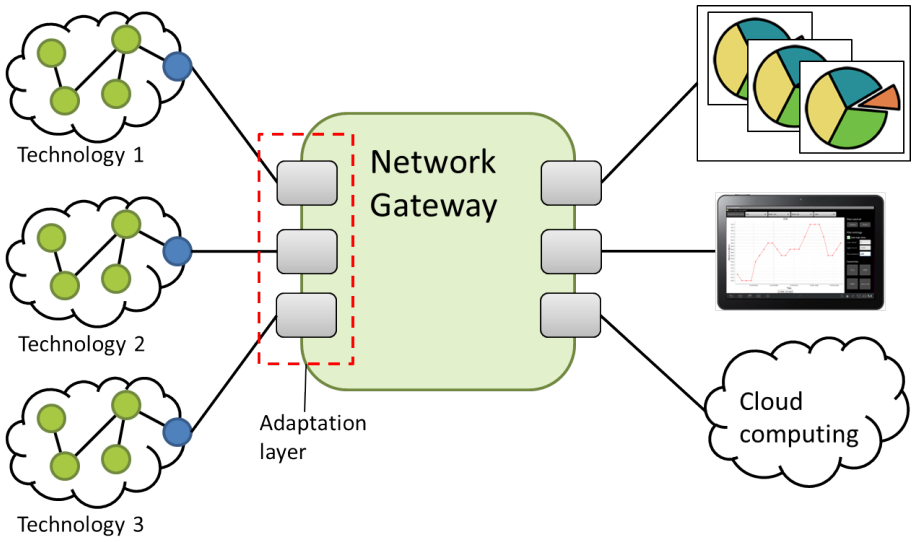

Figure 1.3: Several WSN technologies connected to one network gateway with an adaptation layer.

gateway. The gateway itself does not have any WSN technology specific architectures or functions. This separates the gateway development from WSN integrations.

1.3 Scope of the Thesis

The motivation for this thesis is to integrate different WSN technologies into a gateway so that all of the measurement data can be used without the knowledge of the actual technology or source node of the data. This is done to make the end-user service development easier and quicker. Also, it is possible to change a WSN in an application to another WSN using a different technology, making the application independent of one WSN manufacturer.

This thesis focuses on the adaptation layer between the WSNs and the gateway. Similar adaptation layer can also be used between the end-user services and the gateway but that is out of the scope of this thesis.

In this thesis an adaptation layer is designed and prototyped. To verify the practicality and versatility of the adaptation layer, adapters are implemented for three different types of WSN applications: home automation, biomedical measurement and environmental monitoring WSNs. The implementation of the adaptation layer is tested in real life applications. Generic plotters of measurement data are also used to evaluate the benefits. Requirements for creating an adapter for any WSN technology are explained. All adapters were implemented in Java or C++, the gateway was run on an embedded PC.

The work done in this thesis is a part of Distributed interoperable wireless surveillance, monitoring, and alarm system (WAS) project at Tampere University of Technology (TUT). The WAS project developes a WSN network architecture which is based on heterogeneity, hierarchical levels of intelligence and novel wake-up concept. The gateway used in this thesis is an existing software, called Generic WSN Gateway (GWG). A WSN interface collection, WSN OpenAPI, is also adapted in the thesis. GWG and WSN OpenAPI are both developed at TUT by the TUTWSN research group and are part of the WAS project. This thesis does not focus on the existing GWG development, only on the WSN adaptation layer.

1.4 Outline

This thesis is organized as follows. In Chapter 2, different WSN technologies are introduced to explore the diversity of available WSNs. Chapter 3 introduces related

work to this thesis. The requirements and design issues of adapter development are presented in Chapter 4. Chapter 5 presents the actual implementation and testing work done and explains the made WSN technology choices and compromises in adapters. Result and evaluation is presented in Chapter 6. Chapter 7 concludes the thesis.

2. WSN TECHNOLOGIES

To understand the requirements and challenges of integrating WSN technologies to a gateway, different WSN technologies are presented in this Chapter. From the presented technologies, Bluetooth, ZigBee and Z-Wave are introduced more closely and are used in the actual adapter prototypes of the thesis.

2.1 Technological Diversity of WSNs

There are different WSN technologies available for varying application areas. Technologies have different features and abilities. Features such as frequency band, data rate, power usage and network topology vary between technologies [6].

For building and home automation there are technologies such as ZigBee [12], Z-Wave [13], Wavenis [14] and EnOcean [15]. Bluetooth [16] and ANT [17] are technologies targeted at smaller and simpler one hop WSNs such as mobile phone accessories and heart rate belts. For industrial use there are WSN technologies such as WirelessHART [18] and ISA100 [19]. These WSNs require good reliability, robustness and small latencies in data transfers. Industrial WSN technologies are usually too expensive for personal use. There are also WSN technologies targeted at specific applications. DASH7 [20] is uses a less popular frequency band of 433MHz which

Table 2.1: WSN technology comparision

WSN Technology	Freq. band (MHz)	Data rate (kbps)
ZigBee	868, 915, 2400	20-250
Z-Wave	865, 915	40
Wavenis	422, 868, 915	4.8-100
EnOcean	315, 868	125
Bluetooth 2.0	2400	3100
ANT	2400	1000
WirelessHART	2400	250
ISA100	2400	250
DASH7	433	27.8
RuBee	0.131	1.2

makes it a good choice for long range applications. RuBee [21] has a small form factor and very low power usage but in contrast to these it also has a low data rate and short radio range. Frequency bands and data rates of these WSN technologies are displayed in Table 2.1.

Bluetooth, ZigBee and Z-Wave were selected for integration to the gateway in this thesis. Many biomedical sensors use Bluetooth so it was selected. Using Bluetooth also makes it possible to run the adapter software on any device with Bluetooth support. ZigBee was selected because it is one of the most used WSN technologies [22]. Z-Wave was selected because of the wide range of available devices. The selected technologies have different features and abilities, and are meant for different applications, proving the versatility of the adaptation layer.

2.2 Bluetooth

Bluetooth is a proprietary open wireless technology standard for short range communications [16]. It was originally created by Ericsson in 1994. Although Bluetooth is not originally meant for WSNs it can be used in simple networks. Bluetooth defines two roles for devices in a network: a master and a slave. A master can have up to seven slaves it can transfer data with. Data can not be routed from other nodes, connection are simple peer to peer links.

Bluetooth also supports networks called *piconets* [23]. Piconets add the possibility that slaves can communicate with each other trough the master. Piconets can be

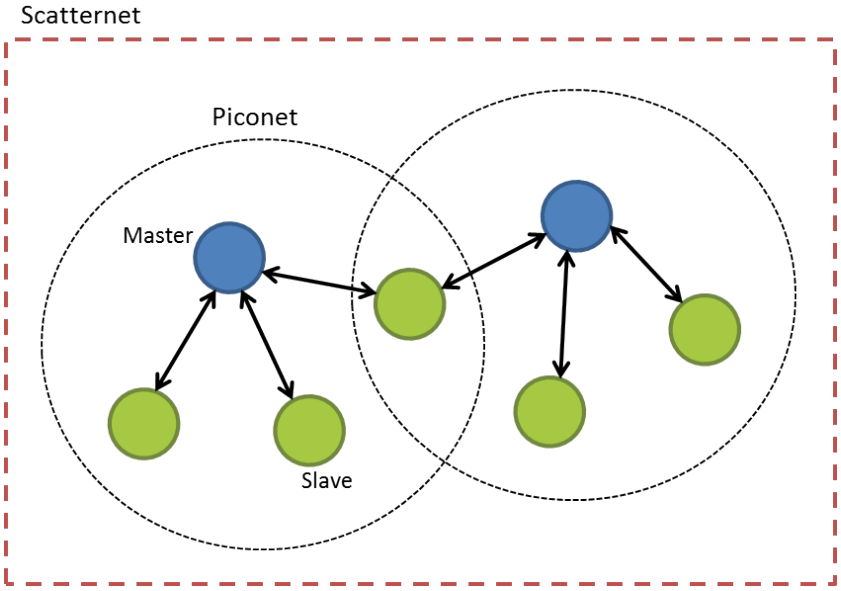

Figure 2.1: Bluetooth device roles, piconet and scatternet topologies.

interconnected to form a larger network called *scatternet*. This can be achieved when a member, either a slave or a master, of one piconet acts as a slave in another piconet. Bluetooth roles, piconet and scatternet are illustrated in Figure 2.1.

Bluetooth operates on the radio band of 2.4 GHz [24]. There are three different transmitter classes used in Bluetooth devices, they are introduced in Table 2.2. Bluetooth version 2.1 has a nominal datarate of 3.1 MBit/s, with a practical rate of 2.1 Mbit/s. These rates allow Bluetooth to be used with simple short-range networks with larger data amounts. Bluetooth version 4.0 introduces a subset called Bluetooth Low Energy (BLE) with a new protocol stack for quick forming of simple links. BLE is meant for low power applications such as wireless sensor networks.

Table 2.2: Bluetooth classes

Class	Max power (mW)	Max power (dBm)	Range (m)
Class 1	100	20	100
Class 2	2,5	4	10
Class 3	1	0	5

Bluetooth has a group of profiles, which are specifications regarding certain applications [24]. Each profile defines certain services which a device must implement to be compatible with the profile. All devices must be compatible with some subset of the Bluetooth profiles. Bluetooh Serial Port Profile (SPP) is a profile that emulates a serial cable to provide simple data handling or substitute an existing RS-232 connection. SPP is often used in Bluetooth nodes to receive the measurement data on the master device.

Bluetooth networks do not use such network gateways as introduced in the WSN architecture of this thesis. To access data of Bluetooth devices, the accessing device has to have a Bluetooth radio and act as the Bluetooth master or slave in a network. The role of the accessing device depends on the implementation of the Bluetooth nodes.

2.3 ZigBee

ZigBee is a specification for a set of high level communication protocols based on the IEEE 802.15.4 [25] standard. The specifications are published by the ZigBee Alliance [12]. Version 1.0 of the specification was announced available on June 13th 2005. ZigBee is a registered trademark, not a single standard. ZigBee is free for non-commercial use, but to use ZigBee in commercial products the manufacturer needs to pay an annual fee.

Figure 2.2: Different products using ZigBee. Universal remote control (left)[33], home automation control center (middle)[34], WSN development kit node (right)[27].

ZigBee was originally meant for applications that require short-range transfer of data at low rates and long battery life, such as wireless light switches, industrial control and home automation [6]. Some example products using ZigBee are presented in Figure 2.2. The development of ZigBee was originally started because WLAN and Bluetooth were not considered adequate for these use areas. The alliance publishes *application profiles* that allow multiple manufacturers to create compatible products. Application profiles define certain properties and functions the device has to implement to use the profile. The current released application profiles are for home automation, energy related usage, telecommunications, health care and remote controlling. The ZigBee specification [26] is intended to be simpler and less expensive compared to other similar technologies. ZigBee also has a basic security model for encrypted and secure communications. The devices operate in radio bands at 868 MHz in Europe, 915 MHz in the USA and Australia and 2,4G Hz worldwide. The data rates vary from 20 to 250 kilobits/second. ZigBee supports star, tree and mesh topologies. In a ZigBee network there are nodes with three different roles, also illustrated in Figure 2.3:

- ZigBee Coordinator (ZC) - Forms the root of the network tree and stores information about the network. There is only one coordinator in each ZigBee network. Coordinators can also form bridges to other networks.
- ZigBee Router (ZR) - Works as an intermediate router, passing data onwards from other devices.
- ZigBee End Device (ZED) - Communicates only with a parent node without routing data from other devices. ZED nodes are asleep most of the time.

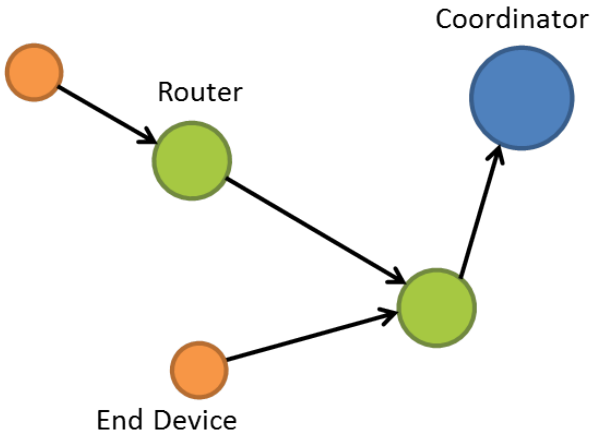

Figure 2.3: ZigBee network terminology.

For ZigBee networks there are network gateways available that provide the measurement data with Application Programming Interfaces (API) with protocols such as Simple Object Access Protocol (SOAP) or REpresentational State Transfer (REST) [12]. The interfaces and protocols vary between manufacturers. The measurement data can also be access straight from a device acting as the ZigBee Coordinator. For example, there are USB devices available, which can be read with a PC.

2.4 Z-Wave

Z-Wave is a wireless communication protocol designed for home automation [13]. It is targeted at residential and light commercial environments. The protocol is optimized for reliable, low-latency communication of small data packets, such as remote controlling. Z-Wave was developed by a Danish startup called Zen-Sys, which was later acquired by Sigma Designs [28]. Z-Wave is supported by over 160 manufacturers worldwide and there are over 600 different products available. Some Z-Wave products are presented in Figure 2.4. The standard is only available to Sigma Designs customers under a non-disclosure agreement, but there is an unofficial open source library called openzwave in development [29]. Z-Wave operates at 868 MHz and 900 MHz frequency bands [30]. There are two bandwidths available for use: 9600 bit/s and 40 kbit/s. The range of the wireless links are usually around 30 meters in open air conditions [31, 32]. Most commercial Z-Wave devices are available to private persons and are meant for use at homes. The devices vary from remote controllable power switches and energy meters to flood and motion detectors.

Z-Wave uses a source routed mesh topology. This means that there are one or more master controllers on the network that control the routing and security. Nodes can also route other nodes' data so the actual usage range can be larger than the single

Figure 2.4: Z-Wave devices.

radio link range. Routing devices can not sleep and therefore battery operated devices are not meant to be used as routers. Z-Wave devices sleep most of the time and wake up at given intervals to send measurements. There can also be event type measurements on nodes, for example on the motion detector. Events are sent right away. A Z-Wave network can consist of up to 232 devices. If more devices are needed, networks can be bridged. To form a Z-Wave network, the node devices have to be manually paired with a gateway device. This is called *inclusion* in Z-Wave terminology. This is usually done with a simple button pressing sequence on the devices. A node can be paired to only one gateway device at a time.

Since Z-Wave is a closed standard, it does not provide any official open interface for accessing the data. Commercial gateway products can be used to control the nodes and create automation scenarios. Openzwave provides an unofficial API that can be used with compatible USB devices. The USB devices can be used to read the measurement data and handle the nodes from a PC.

3. RELATED WORK

This Chapter introduces related work to this thesis. Existing software of the WAS project closely related to this thesis are introduced at the end of this Chapter.

Many middleware solutions for abstracting WSNs from the end-user applications have been proposed but only a small number of them contain architectures for using multiple different WSN technologies together. Most closely related of the proposed architectures are introduced below.

3.1 SeNsIM framework

SeNsIM is a query based framework that provides an architectural and data model for integrating WSNs [35]. The architecture consists from four layers:

- An application or user layer submits queries and elaborates the retrieved data from the mediator layer..
- A mediator layer formats and forwards queries to wanted wrappers.
- A wrapper layer contains WSN specific wrappers for extracting and managing WSN information and data.
- The sensor system layer contains the WSNs with or without middleware or operating systems.

A WSN can be connected to SeNsIM by creating a wrapper for it. Information about the WSN is stored in eXtensible Markup Language (XML) format in the wrapper. The wrapper and the mediator form a connection with this information. After this, user applications can query information about WSNs connected to the architecture and make data queries from wanted WSNs. All data is accessed by queries, data can not be received real time.

There is an implementation of SeNsIM containing a wrapper component for TinyDB based WSNs and a graphical UI for queries. The implementation is written in Java.

3.2 Intelligent Bridge

Ahn and Chong propose an intelligent bridge as an architecture for message exchanging between WSNs [36]. In their architecture a server computer is used as a bridge between two or more networks. Messages are sent from the WSN gateways to the bridge in XML format using SOAP as transmission protocol. The bridge has rules on which it acts upon when it receives messages. Their implementation of the proposal includes message types for query messages, measurement messages, events and actuator control. The messages have to contain information of the destination WSN.

The intelligent bridge does not contain a network gateway. Measurements are only transferred between WSNs. This makes it hard to use the measurement data in end-user services. In this approach a WSN must be aware of other WSNs to send them messages. The rule set in the bridge can also grow large when there are more than two WSNs connected.

3.3 Global Sensor Network

Global Sensor Network (GSN) is a proposed middleware that provides a uniform platform for integrating sensor networks [37]. The main idea in GSN is virtual sensors. A virtual sensor is an abstraction of any kind of data producer, such as a real sensor, a cell phone or any combination of different virtual sensors. All virtual sensors must be described in XML. Virtual sensors use a wrapper to communicate with GSN API to store measurements in a Structured Query Language (SQL) database. Measurements from all sources can then be queried from the database. GSN does not support real time data acquiring, all measurements are first stored in the database.

3.4 Other Proposals

Two similar proposals are introduces below. These proposals are meant for WSNs but are Internet based, and therefore meant for different applications.

IrisNet is web infrastructure for integrating WSNs [38]. The architecture consists of data sources and data organizers. The organizers store the data from the sources into a hierarchial, distributed XML database which can be accessed with XPath queries.

Hourglass is an Internet-based infrastructure for connecting sensors, services and

applications [39]. Measurement data from the sensors is routed to the client applications and the underlying sensor networks are completely hidden. WSNs are connected to the Hourglass infrastructure with proxy services. Hourglass supports in-network services such as filtering, aggregation, compression, and buffering stream data between source and destination. Third party services can also be deployed and used in the network.

3.5 Proposal Shortages

The key features for a multi-technology WSN architecture introduced in this thesis are:

- Possibility to acquire measurements real time
- Works both locally and over the Internet.
- All measurement data can be accessed and used similarly regardless of the source technology.
- Has to be implementable.

All of the introduced proposals lack at least one key features of a multi-technology WSN architecture or are meant for other purposes. The intelligent bridge does not use an actual network gateway software, making end-user services hard to implement and scalability low. GSN and IrisNet support only acquiring messages from a database, with no support for using measurement data in real time. Constraining the infrastructure to use the Internet or a specific transmission protocol also limits the versatility. The introduced proposals do not present any models or instructions for the actual implementation but give a more overall view of the complete architecture.

This thesis uses an architecture that has all the mentioned key features. It supports real time measurements and provides interfaces for end-user services. It is not tied to any specific transmission protocols or computer architectures. The adaptation layer makes WSN technology integration to the gateway quick and straightforward. The architecture scales from just few nodes to thousands of nodes. This thesis also presents the actual implementation of three adapters.

3.6 Generic WSN Gateway

Generic WSN Gateway (GWG) is a WSN gateway architecture developed at TUT [40]. An implementation of GWG is used in this thesis as the gateway software. It is designed to be WSN technology independent. Main functions of the GWG is to receive data from different WSNs, store measurements in a database and pass measurements forward to multiple end-user services. GWG architecture is illustrated in Figure 3.1. It consists from the actual gateway and the adaption layers.

GWG uses internally an interface definition called WSN OpenAPI [40]. Technology specific message formats have to be converted into WSN OpenAPI in the adaptation layer. The adaptation layer hides the underlying WSN technologies from the GWG. This way the GWG can use all different WSNs similarly. The adaptation layer consists of several technology specific adapters. In addition to the WSN OpenAPI conversion, an adapter may also implement missing functionalities to a WSN. For example an end-user service can subscribe only wanted measurement quantities from the network. If the network itself does not support this functionality, the adapter can implement it by filtering out unwanted quantities. This thesis focuses on the adaptation layer between WSNs and the gateway and the term adaptation layer is used to refer to this layer later in this thesis.

Addition to the actual gateway core, GWG can also have a database for storing

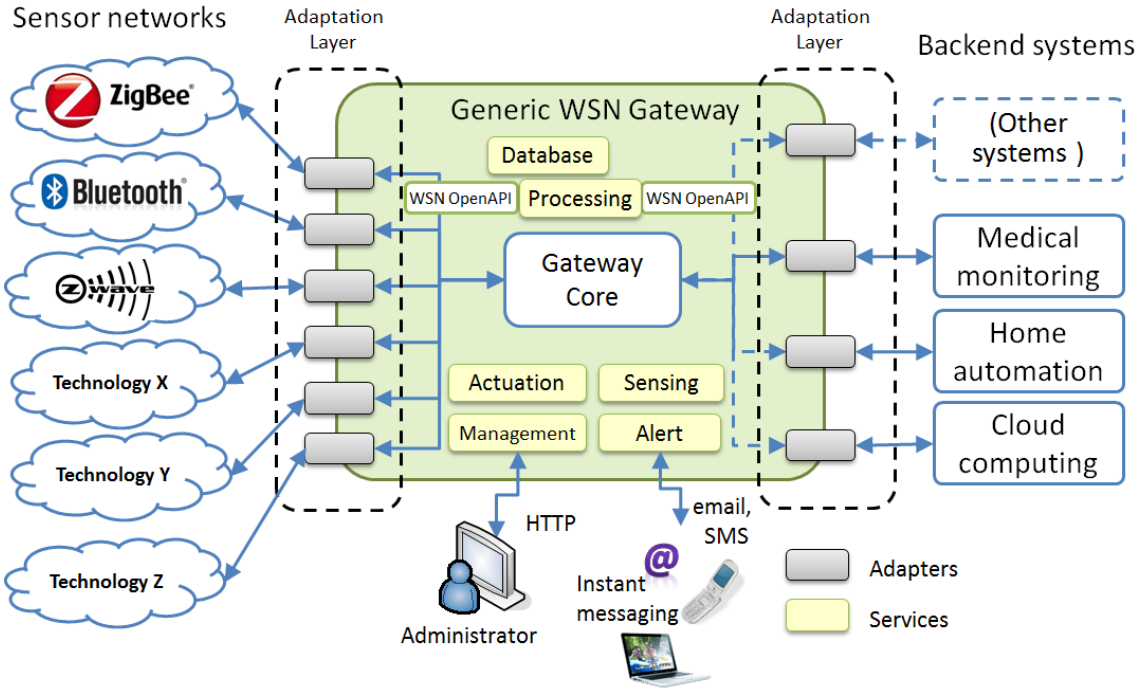

Figure 3.1: Generic WSN Gateway architecture.

measurements. It can also process data, for example calculate actual acceleration from X, Y and Z components. The processing can also be implemented in adapters and sent as new quantities. Data can be transferred from the WSNs to the end-user services but also to the other way. Actuators in WSNs can be controlled from the end-user services, administration interface or other WSNs. GWG has also features for sending alerts for measurements that exceed set boundaries, for example too high temperature.

3.6.1 Generic WSN Gateway Configurations

There are four different versions of GWG for different applications. These versions are Tiny, Basic, BasicWeb and Full, as illustrated in Figure 3.2. Tiny, Basic and BasicWeb GWGs use static configuration files for deploying services and applications. These are done statically at the startup of the software. Full version uses JBoss for dynamic deployment which can be at runtime. Tiny and BasicWeb were used in the integration phases of this thesis.

- Tiny consists of only the core features of the GWG. It has all the basic applications and services needed for a basic gateway. All configuration is done with configuration files and loaded on startup.

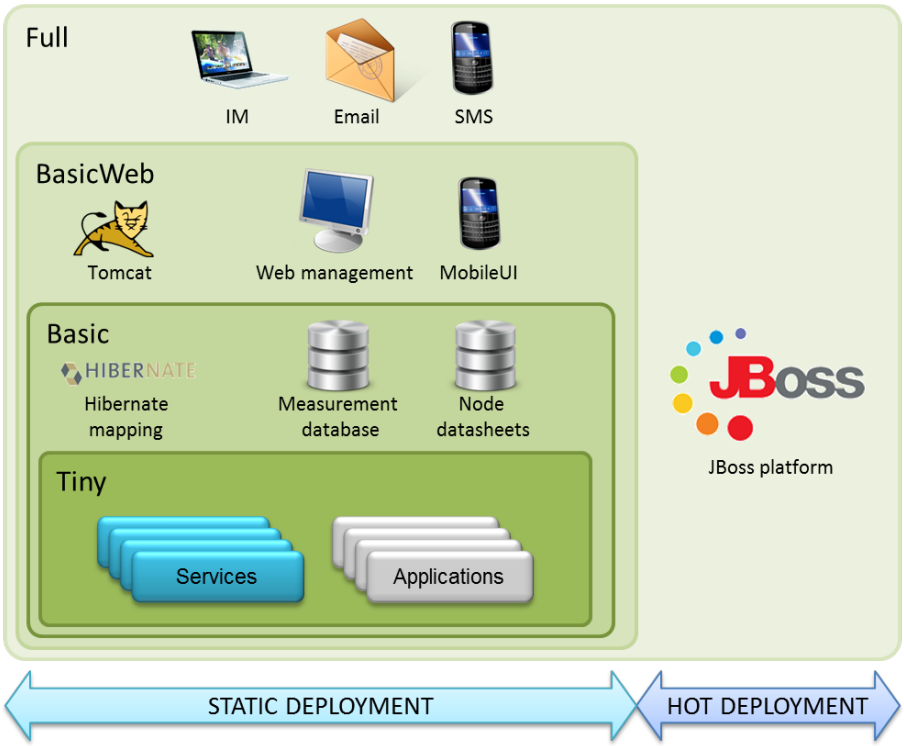

Figure 3.2: Generic WSN Gateway configurations.

- Basic adds a database for measurements and node datasheets to Tiny version. Configurations are also read from a database and can be changed during operation.
- BasicWeb adds a web user-interface (UI) to Basic version. The UI can be used for configuring the gateway and node mappings.
- Full adds support for instant messaging (IM), email and SMS alerts. Full version is a work-in-progress.

3.7 WSN OpenAPI

WSN OpenAPI is a collection of interfaces for WSN communications [40]. It is designed to be a compact, unified and technology independent. WSN OpenAPI messages use XML or Comma-Separated Values (CSV) formats. The XML format is used when there is enough bandwidth to transfer these messages and the receiver software can handle such amount of data. CSV format is more compact and reduces data traffic and is therefore suitable for resource limited receivers. On the downside, the CSV format contains no meta-data. In the adapter development of this thesis only the XML format was used.

WSN OpenAPI provides different types of messages as listed in Table 3.1. Benefit of having different types of messages is increased modularity. Only needed types can be implemented in an application to reduce the amount of work and to make the software lighter. The relevant message types for this thesis are Authentication and Capability Format (ACF), Sensor Information Data Format (SIDF) and Node Actuator and Sensor Control (NASC).

Table 3.1: WSN OpenAPI message types

Type	Purpose
ACF	Authentication with gateway and capabilities requests.
MEDF	Accessing meta-data of nodes.
NASC	Sending of commands to nodes.
NMF	Configuring of the network itself.
SADF	Accessing measurement data from a data archive.
SIDF	Transport of measurement data.

3.7.1 Authentication and Capability Format (ACF)

The Authentication and Capability Format (ACF) is used for authenticating clients to the gateway. It contains only the mandatory features to establish an authentication. A client has to authenticate to the gateway, otherwise all messages are ignored.

ACF supports three methods for authentication:

- none - formal authentication is not required
- password - authentication is done with username and password
- challenge - authentication is done with challenge-response method

Password method is the most common and it was used in this thesis. The username can be given access rights to defined networks, for example send and receive SIDF messages to and from a certain network. The gateway responds to the client whether the authentication was accepted or not. An example ACF message in XML format is presented in Listing 3.1.

```
1 <AuthenticationRequest xmlns="urn:ietf:params:xml:ns:acf"
    method="password" responseFormat="XML" version="1.2">
2     <Parameter name="username" value="testuser"/>
3     <Parameter name="password" value="testpassword"/>
4 </AuthenticationRequest>
```

Listing 3.1: ACF message in XML format

3.7.2 Node Actuator and Sensor Control (NASC)

The Node Actuator and Sensor Control (NASC) messages are used for controlling actuator nodes in the networks. A NASC message can contain commands for only one node because the messages can be transported directly to the nodes. An example NASC message is presented in Listing 3.2.

```
1 <NASC xmlns="urn:wsn-openapi:xml:ns:nasc" messageId="1"
    version="1.0">
2     <Network id="1">
3         <Node id="2">
4             <Actuator id="Power Switch">ON</Actuator>
5         </Node>
6     </Network>
7 </NASC>
```

Listing 3.2: NASC message in XML format

3.7.3 Sensor Information Data Format (SIDF)

Sensor Information Data Format (SIDF) is a message format designed to transport measurement data between a WSN and a gateway or between a gateway and end-user services. One SIDF message can only be used to transport measurements from a single node. Several measurements can be buffered and transferred in one SIDF message to reduce the amount of messages. An example SIDF message in XML and CSV is presented in Listing 3.3 and Listing 3.4. The example is intended to help reading, in real use indentations are left out.

```
1 <SIDF xmlns="urn:ietf:params:xml:ns:sidf" version="1.4">
2     <Network id="1">
3         <Node id="2">
4             <Sensor id="3">
5                 <Measurement quantity="Temperature" unit="C"
                      time="2012-04-20T013:05:25+00:00"
                      timeOffset="043">
6                      <Component id="value">19.5</Component>
7                 </Measurement>
8             </Sensor>
9         </Node>
10    </Network>
11 </SIDF>
```

Listing 3.3: SIDF message in XML format

```
1 SIDF;1.4;1;
2 "1";;"2";;"3";;2012-04-20T013:05:25+00:00;043;;19.5;;;
```

Listing 3.4: SIDF message in CSV format

A client can request to receive SIDF messages from the GWG. These request, called subscriptions later in this thesis, can be for all network or from specific networks, nodes and measurements. An example subscription in XML format is presented in Listing 3.5.

```
1 <Request xmlns="urn:ietf:params:xml:ns:sidf" version="1.4">
2   <Network id="1">
3     <Node id="2"/>
4     <Node id="Temperature sensor"/>
5   </Network>
6 </Request>
```

Listing 3.5: SIDF Request for measurements in XML format

4. ADAPTATION LAYER DESIGN

This Chapter presents the requirements and features of an adaptation layer and the adapters designed in this thesis.

4.1 Adapter Requirements

The basic idea of an adapter is to work as a bridge between a WSN and a network gateway. The key requirements of an adapter are:

- Identify and separate all physical devices and map them to their logical counterparts.
- Connect, initialize and upkeep the WSN.
- Receive measurements from the WSN and send user commands to the WSN.
- Transform all WSN specific messages to WSN OpenAPI and vice versa. This includes transforming user commands to control the WSN.
- Form and upkeep connection to GWG.
- Implement required parts of WSN OpenAPI.

The adapter needs interfaces for communicating with the WSN and the GWG. These interfaces can also be one way, for example if there are no actuators in the WSN, no data needs to passed to the WSN. The device at the edge of the WSN, communicating with the adapter, can be a network gateway or a sink node. For the adapter it doe not matter what the device is, the adapter only converts the messages between the formats.

A general adapter model used in this thesis is illustrated in Figure 4.1. The model is divided into three parts: the WSN side, GWG side and the data processing part. The WSN side interface is technology specific. Some technologies may need initializations or more complicated communications trough the interface while some

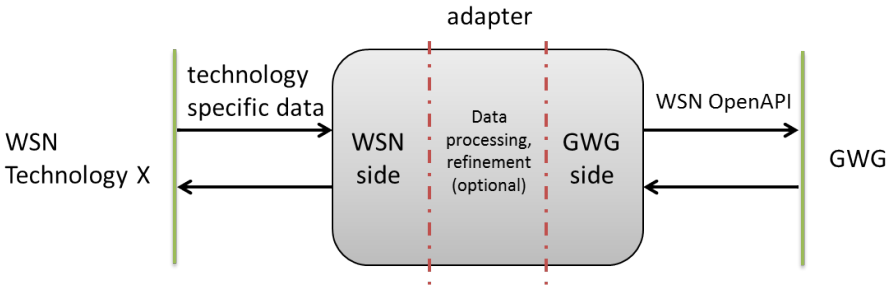

Figure 4.1: General adapter model.

technologies only send data at certain intervals. The GWG side interface needs to implement the WSN OpenAPI requirements. The adapter can implement only the required parts of WSN OpenAPI. For example if only measurements are passed to GWG the adapter only needs to implement ACF for establishing connection to the GWG core and SIDF for the sending measurements. If support for controlling actuators is needed, NASC has to be implemented.

Support for additional features can also be implemented into the adapter. For example, some WSNs support interests which requests for the nodes to send wanted measurements at given intervals. An example sequence diagram of requesting temperature measurements from the nodes every two minutes is illustrated in Figure 4.2. If the WSN does not support interests, the adapter can simulate interest by polling the values at given intervals, as illustrated in Figure 4.3.

The adapter can also process or refine the measurement data. For example, if the WSN does not include measurement units with the actual measurement values, the adapter can add these to the WSN OpenAPI messages. Processing of the data is

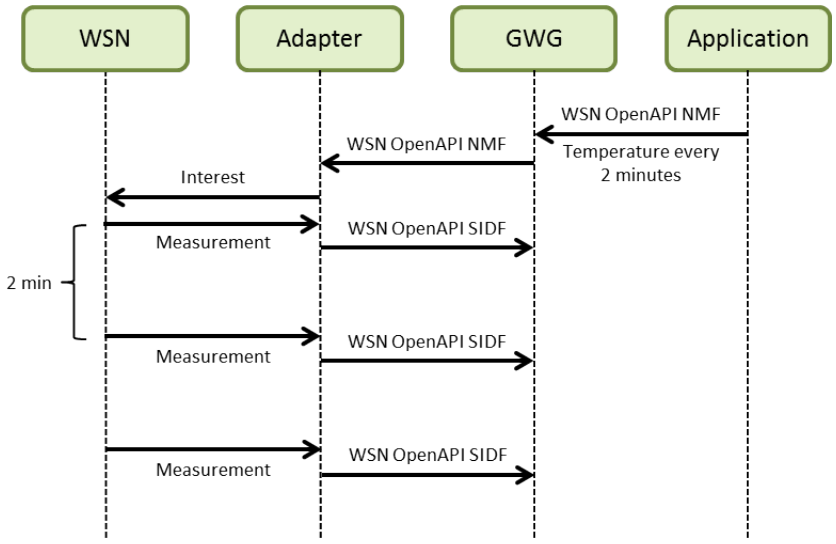

Figure 4.2: Sequence chart of interest based measurements

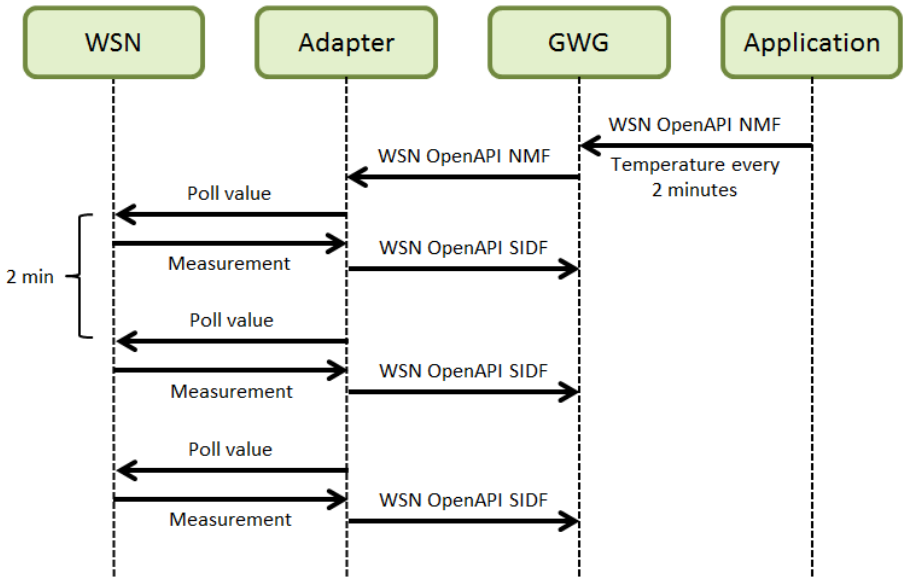

Figure 4.3: Sequence chart of polling based measurements

adapter specific and optional.

For the adapter to be able to do the mapping it has to be aware of all physical devices and the logical counterparts. This is is done either manually or automatically, for example by using the DeviceIDs straight as the logical NodeIDs. There could be a database of the mappings where adapters would lookup the mapping information or the adapters could communicate with each other or the GWG to avoid using the same identifiers. In this this thesis adapters are made as simple as possible to speed up the integration process of a new WSN technology.

4.2 Device Mapping

When connecting multiple WSN technologies together, one major issue is how to uniquely identify each physical device connected to the GWG. Technologies have their own way of identifying devices, usually a unique identifier, referred to as DeviceID in this thesis. This works when using only one WSN but with several WSNs it is insufficient because different WSNs can have devices with the same DeviceID. In this thesis, this is resolved by giving all WSNs a unique technology identifier, TechnologyID. The TechnologyID separates physical networks from each other and the DeviceID identifies the individual physical devices in them. In this chapter the term device refers to a physical device with one or more sensors or actuators.

To make the usage of several WSNs together easier and more flexible the adaptation layer maps the physical devices to logical nodes and networks [40]. Logical networks

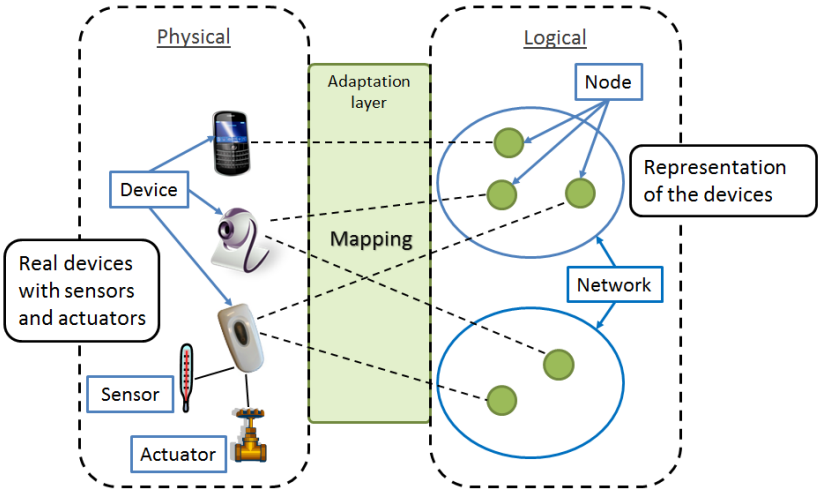

Figure 4.4: Mapping of physical nodes to logical nodes.

are groups of logical nodes. Logical networks and nodes should not be confused with the physical WSNs and devices. In this thesis, one device can be mapped to several logical nodes in different logical networks. The mapping is illustrated in Figure 4.4. Logical nodes and networks are given unique identifiers, NodeID and NetworkID. Different networks can have nodes with same NodeIDs since the NetworkIDs are different.

Mapping enables end-user services to use measurement data without specifying which device actually produced the data and what WSN it is from. Also actuators can be controlled without the knowledge of what technology it uses. The end-user services have access only to the logical networks. This makes end-user service development easier and more efficient.

4.3 Adapter Location

The adapter can be located in three different places as illustrated in Figure 4.5. The location depends on the WSN technology and the devices used. The adapter locations are in the WSNs sink node or network gateway, in the GWG or between the sink node and GWG. In all cases the responsibilities and requirements for the adapter are the same.

If the adapter is implemented in the sink node, it works close to the network and enables tight cooperation with the WSN. Implementing the adapter in the sink node is not always possible because modifying the node's software is not possible in all WSNs. WSN development kits and more user-tailorable WSNs usually provide access to the sink node software. The Bluetooth adapter developed in this thesis

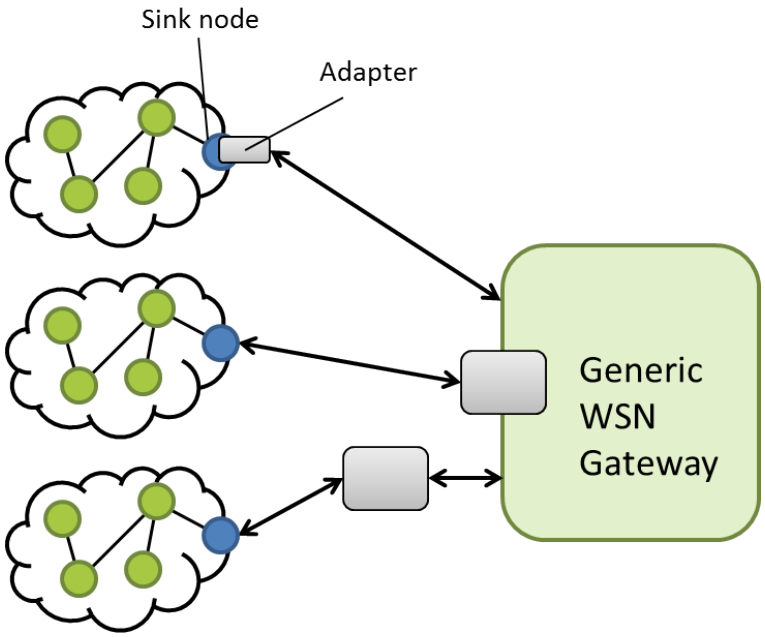

Figure 4.5: Different locations for adapters.

represents this approach.

The GWG supports implementation of applications to create new features. The adapter can be implemented to the GWG as such application. This way the adapter is a more integrated part of the gateway and not a standalone component. This approach is more dependent on the GWG but also enables a more efficient data exchange between the adapter and the GWG. On the downside, major changes in the GWG core may break adapter functionality.

If the adapter is created as an independent component between the WSN and the GWG it can be developed independently and can be easily altered for different use cases. It does not require any customization to the WSN software or the GWG. On the downside, data exchange rates may be lower because of the loose integration to both the WSN and GWG.

4.4 Adaptation Process

This section provides general instructions on how to create an adapter for any WSN technology. Depending on the technology to implement, there are different number of phases in the software implementation. The software to be developed can be divided into two main parts: the WSN side software and the adapter side software, as illustrated in Figure 4.6. The WSN side software consists of the node softwares to send measurements between the nodes and the sink node software to send mea-

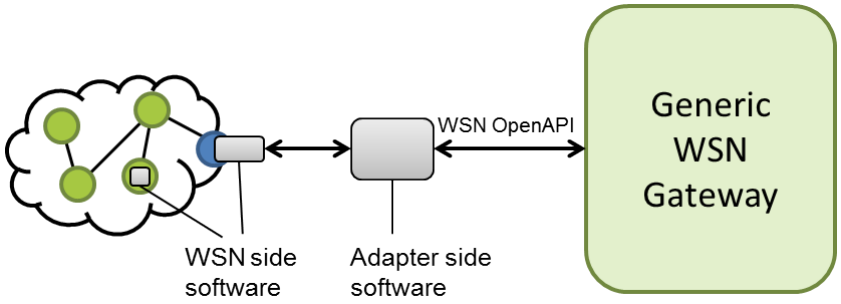

Figure 4.6: Components of an adapter

surement to the adapter. The adapter side software is the actual adapter. These two sides must be developed closely together and can be thought as one adapter. In some WSNs, there are no existing reference software to send measurements between the nodes or onwards from the sink node. The nodes may only contain software to handle the node platform and radio. In these cases node software for sending measurements has to be developed. If the nodes already contain software only the adapter side software is developed. The idea is same in all cases, all the measurements are sent from the WSN side to the adapter side, transformed to WSN OpenAPI and sent onwards to the GWG.

4.4.1 WSN Side Software

The first phase, if needed, is to create the software for all the nodes or just the sink node depending on the technology. First the packet format of the packets between the nodes and the adapter software has to be decided. The used technologies may limit the possible formats. A good general and simple packet format is to send a start sequence, the source node id, timestamp and the measurement values coupled with the quantity of the measurement. The quantities can be coded with for example numeric values which are later decoded at the adapter software. This is to minimize the amount of data to be sent.

An example simple packet format with example byte sizes is illustrated in Table 4.1. In this packet the quantities are coded with simple numeric values. For example the quantity value 0x01 could represent temperature and 0x02 could be humidity. The coding and byte lengths have to be decided and kept the same in both the WSN side node software and the adapter software. The packet can also include units for the quantities if they are available.

First the measurement data has to be acquired from the sensors. The methods of acquiring the measurement values and node ids on the nodes vary greatly between

Table 4.1: Example simple packet format.

Meaning	Start	Timestamp	Node	Quantity	Value	Quantity	Value
Value	0xFF	0x4FBC9B4E	0x12	0x01	0x0016	0x02	0x55
Size	1B	4B	1B	1B	2B	1B	2B

different technologies. When these are acquired they are packaged into the decided packet format. The packet is then sent to the adapter side software. The methods for sending the packet also greatly vary. Often the manufacturer provides ready made APIs for common operations.

4.4.2 Adapter Software

The actual adapter side software receives the measurements from the sink node, acting as the WSN side part of the adapter. This is often done by reading a serial data stream. The software has to communicate with the physical receiver device to read the measurements. After a packet is received it has to be transformed into WSN OpenAPI and optional processing done to measurement values. This is the data processing part of the adapter. After the conversion, the adapter sends the WSN OpenAPI messages to GWG, acting as the GWG side part of the adapter. The adapter software has to implement at least ACF to authenticate with GWG and SIDF or NASC to handle measurement data.

5. ADAPTER IMPLEMENTATIONS

Prototype adapters for ZigBee, Z-Wave and Bluetooth were created in this thesis to test the designed adapter model. It approves that multiple technologies can be used together seamlessly. The Bluetooth adapter also contained a local UI for plotting the measurements.

An Android software, called TUT RemotePlot, for plotting measurements from the WSNs was also implemented to test the adapters. An existing PC software, called WSN Monitor, was also used for plotting the measurements.

The implemented architecture is illustrated in Figure 5.1. Generic WSN Gateway and the PC plotter were existing software from the project, all other components were implemented as part of this thesis.

This Chapter presents the implemented adapters and softwares. It also introduces the tools and platform used for implementation.

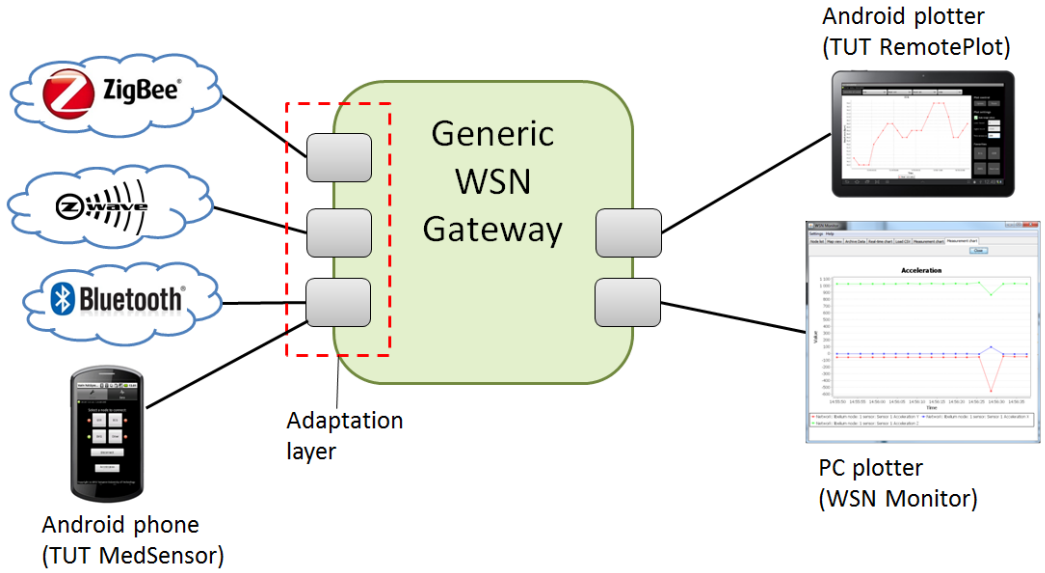

Figure 5.1: The implemented architecture.

5.1 Development Tools and Platform

Tools used in this work are presented in Table 5.1. Most of the Java and C++ code were written with Eclipse [41]. The thesis itself was written with TeXworks which is a part of MiKTeX [42].

Table 5.1: Tools and platforms used in this work.

Tool	Version
Android	2.2 and 3.0
Android Software Development Kit	r20
Waspmote IDE	0.2

5.2 Android

Android is a Linux-based open-source operating system for mobile devices [43]. It is mostly used in smartphones and tablet computers. The development is done by the Open Handset Alliance which is led by Google. Android has been the best selling smartphone operating systems since Q4 of the year 2010 [44]. The latest version is 4.1 which was released in July 2012.

5.2.1 Android Software Development Kit

The Android Software Development Kit (SDK) is a set of tools for developing applications for Android [45]. It includes all essential tools including libraries, compiler, documentation and tutorials. It can be run on Windows, Linux and Mac OS X operating systems. The suggested integrated development environment (IDE) is Eclipse, which was also used in this thesis. Developers can also use a text editor of their choice and command line tools.

The SDK can be downloaded from Google for free. Application development for Android is free but putting the applications to Android's online application store Google Play, the developer has to pay a one time developer fee. The least expensive Android phones can be bought for under 100 euros. The SDK includes an emulator for testing the applications so a physical smartphone is not mandatory for developing. These make Android application development inexpensive and easily approachable.

5.2.2 Android applications

Android applications are usually written in Java language. It is also possible to develop applications using extensions for C and C++ or a visual environment for novice programmers. All Android applications developed in this thesis are written in Java.

Android applications compose from building blocks called application components [45]. There are four different types of application components. Each type has a distinct purpose and lifecycle that defines how the component is created and destroyed. The four types and their roles are:

- Activities - A single screen with an user interface. User interface components can be only accessed from the corresponding activity. Activities can be started from anywhere in the applications.
- Services - A component that runs in the background. Services are used for performing long-running operations. A service does not have a user interface and it can not handle them.
- Content providers - Manages data in a shared location. File system, database, online storage or any persistent storage location can be used. Through the content provider any application can query data with proper permissions.
- Broadcast receivers - A component that handles system-wide broadcasts such as an announcement that the screen has turned off or the volume has been increased. Applications can also send broadcasts, for example an announcement that some data has been downloaded and is now available.

5.3 Bluetooth Nodes

Bluetooth nodes are available from many manufactures for varying purposes. All nodes use Bluetooth to send the measurements, but the actual measurement package formats vary between devices. In this thesis, one adapter with device specific parts is created. Because the adapter handles more than one device type and it also handles the Bluetooth connection, the term gateway is more appropriate. The basic idea of it is still the same as with an adapter.

Smartphones have nowadays lots of processing capabilities and multiple radios such as Bluetooth, Wireless Local Area Network (WLAN) and 3G modems. Smartphones

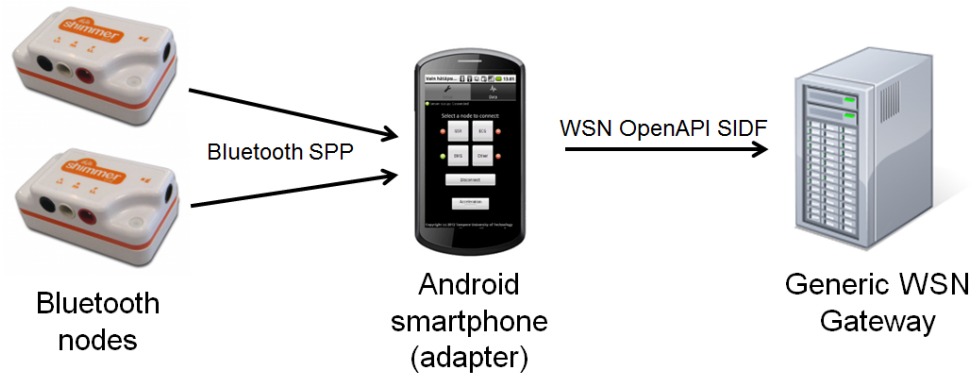

Figure 5.2: Basic idea of the Bluetooth node gateway software.

are also small and lightweight. An Android smartphone is a suitable platform for the Bluetooth adapter since Android provides extensive and easy to use APIs for using the radios.

The adapter bridges the measurements from Bluetooth to either a local network or the Internet with WLAN or 3G and onwards to the GWG. The adapter is implemented as a normal Android application. A feature to plot measurements locally on the smartphone is also implemented in addition to the gateway functions. The basic idea of the setup is illustrated in Figure 5.2.

5.3.1 Android Phone Gateway Software

A prototype mobile gateway software for Bluetooth nodes was created for this thesis. It was used for biomedical sensors and therefore named as TUT MedSensor. MedSensor was designed and implemented to be used with nodes from different manufacturers. The technology specific parts are separated from the core functionality. The basic structure of the application is illustrated in Figure 5.3. The application is divided into four main components: the main activity, chart view service, data handling service and the remote server connection service.

The main activity handles the initialization and overall flow of the application and the graphical user interface. The chart view service is responsible for refining the measurement data to a graphical plot. The data handling service is responsible for communicating with the Bluetooth nodes and transforming the measurement data to WSN Openapi, therefore acting as the WSN side and processing component of the adapter. It passes the data onwards for the chart view service and the server connection service. The server connection service acts as the GWG side of the adapter. TUT MedSensor implements ACF and SIDF types of WSN OpenAPI. It uses an WSN OpenAPI Java library developed at TUT to create the messages and

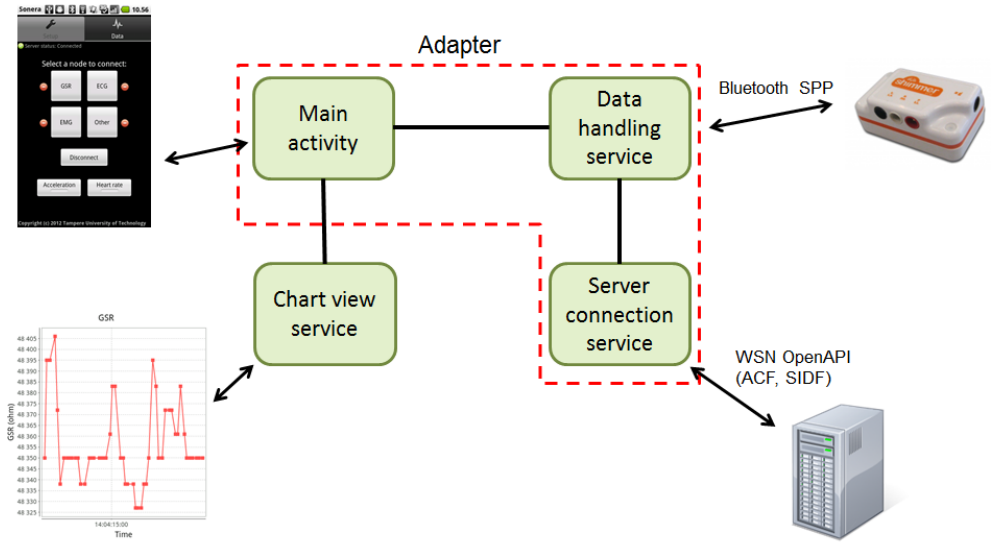

Figure 5.3: TUT MedSensor application functional structure.

form the connection to the GWG.

Figure 5.4 shows the phases done in MedSensor when a new packet is received from a Bluetooth node. After the packet has been received it is parsed by the technology specific component. If the received data needs refinement or transformation, it is done in this phase. After the optional data processing, the message is converted into WSN OpenAPI SIDF message format and then passed to the remote server connection component, which passes the message to a remote GWG. The messages are queued in the service, and sent in a separate thread to avoid becoming a bottleneck in the application, if the connection is slow or unreliable. Especially when 3G

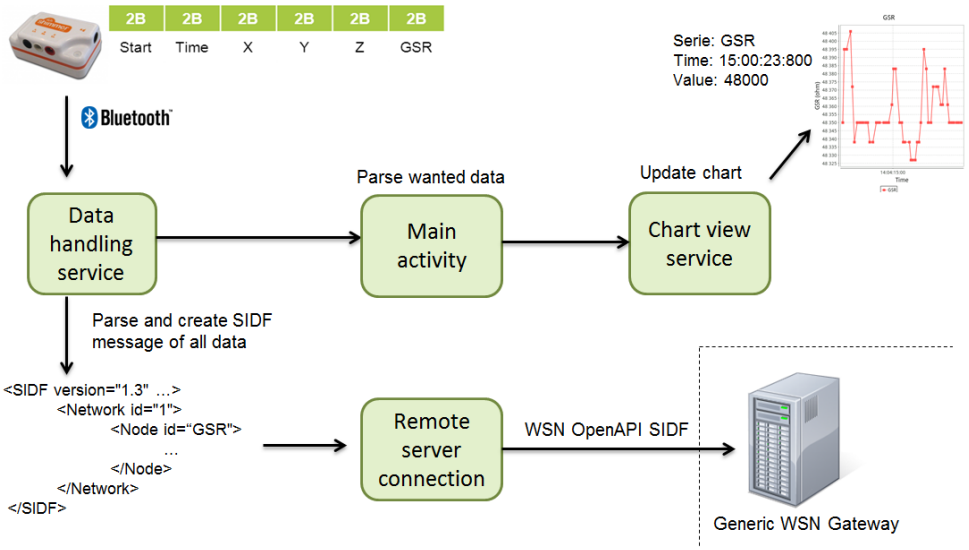

Figure 5.4: TUT MedSensor receiving a new measurement packet.

is used, connection may be unreliable. The measurement packets are also passed to the chart view service through the main activity. Data is then plotted on the graph view in real time.

Figure 5.5 shows the two main screen of the MedSensor user interface. The main screen has customizable quick selection buttons for connecting to Bluetooth nodes. It has buttons for disabling some of the measurement components for increasing the performance. The chart view shows the plot of the current measurement data and selections for the Y-axis value scaling.

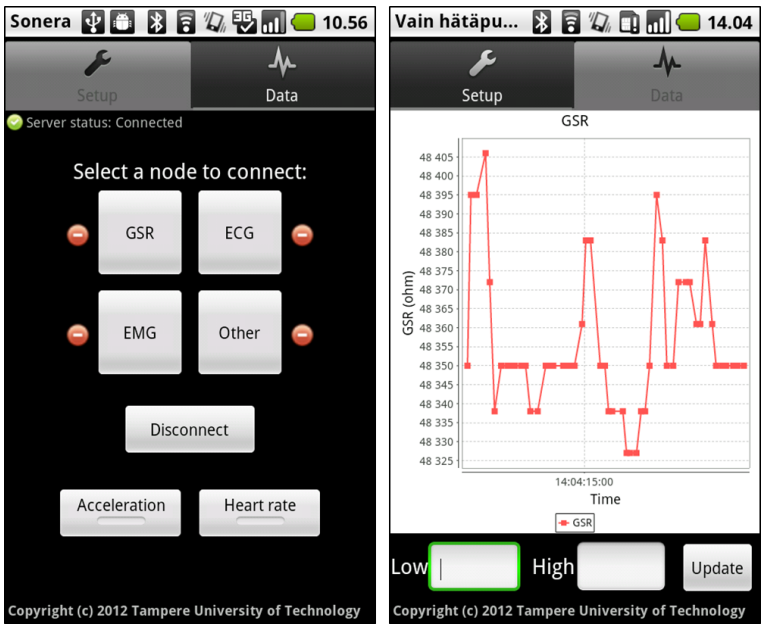

Figure 5.5: TUT MedSensor application graphical user interface.

5.3.2 Biomedical Sensors

There are many manufacturers with nodes for biophysical measurements. Shimmer [46] nodes, shown in Figure 5.6, were selected because of their open source reference implementations and good support for developers. Nodes for measuring galvanic skin response (GSR), electrocardiography (ECG) and electromyography (EMG) were used. Table 5.2 presents technical information of the nodes.

Table 5.2: Shimmer Bluetooth nodes technical information

Clock frequency	8 MHz
Memory (RAM)	10 kB
Memory (Flash)	48 kB
Sample rate	10 Hz - 1k Hz
Battery capacity	450 mAh
Battery life	approx. 1 day
Range	max. 50 m

Shimmer nodes have Bluetooth and 802.15.4 radios, but only the Bluetooth radio was used in this implementation. All nodes contain 3-axis acceleration sensors and other sensors can be connected modularly. Additional sensors are available for measuring GSR, ECG, EMG, rotation, magnetic fields, strain, infrared, temperature, and pressure. The nodes contain a 8 MHz MSP430 processor and they run TinyOS [47], an open source operating system designed for low-power wireless devices, developed by members of the academic world and companies. The software of Shimmer nodes is freely customizable and is written in nesC language, which is an extension to C. Regardless of that, the adapter can not be implemented in the node itself because the GWG does not have support for Bluetooth connections. The measurements have to be bridged to WLAN or 3G.

Figure 5.6: Shimmer Bluetooth nodes. [46]

A reference implementation was used to save time and effort, no changes were needed to the node software. The software sends simple packets, illustrated in Table 5.3 where each actual measurement takes two bytes. The packets contains a start sequence, a timestamp and raw measurement data from the sensors. The start sequence indicates the start of a new packet to simplify the receiving. The timestamp was not used in this implementation as it is only 2 B in length and overflows periodically. Instead, measurements are timestamped in MedSensor as they are received. The manufacturer provides formulas for converting the raw measurement data to sensible values. Since the packets do not include any measurements units, they are added in addition to the data refinement in the data processing component.

Table 5.3: Example Shimmer measurement packet format

2B	2B	2B	2B	2B	2B
Start	Time	AccelX	AccelY	AccelZ	GSR

All nodes include the acceleration data of three axises. EMG and GSR have only one channel for the actual measurement data so the total packet size for them is 12 B. ECG has two channels, leading to a total packet size of 14 B. For example, a sample rate of 50 Hz was used for the ECG node which means a total bandwidth required from the Bluetooth connection is 700 B/s. This bandwidth is low enough for Bluetooth connections. One Shimmer ECG packet with acceleration and two ECG channels converted into WSN OpenAPI takes approximately 600 B. With the sample rate of 50 Hz, a bandwidth of 29 kB/s is required from the GWG connection. This can be achieved in WLAN, but mobile data connections can become a bottleneck. If several nodes are used at the same time the GWG connection quality becomes even more critical.

During the testing of the software and nodes, the mobile phone processing capability also seemed to become a bottleneck. Especially the plotting of data required lots of processing power. Low-end smartphones were noticed not to perform well enough. High end models performed well even with larger data amounts and higher sample rates.

5.3.3 Heart Rate Sensor

Heart rate can be calculated from the ECG signal. This method was first used with the ECG Shimmer node but it turned out to be sensitive to inaccuracies. Shimmer nodes were also uncomfortable to wear, especially in sports. A chest strap with a Bluetooth heart rate sensor meant for only monitoring the heart rate was seen as a

solution to these problems. Zephyr HxM[48] Bluetooth sensor, shown in Figure 5.7, was selected for integration because the manufacturer provides APIs for accessing the data and comprehensive documentation. The HxM is meant for personal use and it is compatible with many smartphone fitness applications. In addition to monitoring the heart rate it also tracks speed and distance, based on steps. According to the manufacturer the sensor has a battery lifetime of 26 hours and it can be recharged from Universal Serial Bus (USB). The belt uses a sampling rate of 1 Hz.

TUT MedSensor was also used with the heart rate belt. Only the technology specific component was added to the existing software and therefore integration of the sensor was fast. The HxM specific packet format is presented in Table 5.4. The packets are 60 B and the actual heart rate, byte 12, takes up only 1 B. Other relevant bytes are the battery charge rate, byte 11, and the movement data, bytes 50 to 54. Bytes 14 to 40 are used to store timestamps of the heart beats. Zephyr also uses two bytes for timestamps. These timestamps can be used to calculate intervals in seconds between consecutive heart beats but they can not be used as a timestamp for the packets. The packets are timestamped in the MedSensor software.

Figure 5.7: Zephyr HxM Bluetooth heart rate sensor. [48]

Table 5.4: Zephyr HxM packet format

Byte	Content
0	STX
1	0x26
2	55
3	Firmware ID
5	Firmware version
7	Hardware ID
9	Hardware version
11	Battery charge percent
12	Heart rate
13	Heart rate number
14	Heart rate timestamp #1 (newest)
16	Heart rate timestamp #2
18	Heart rate timestamp #3
20	Heart rate timestamp #4
22	Heart rate timestamp #5
24	Heart rate timestamp #6
26	Heart rate timestamp #7
28	Heart rate timestamp #8
30	Heart rate timestamp #9
32	Heart rate timestamp #10
34	Heart rate timestamp #11
36	Heart rate timestamp #12
38	Heart rate timestamp #13
40	Heart rate timestamp #14
42	Heart rate timestamp #15 (oldest)
44	Reserved
46	Reserved
48	Reserved
50	Distance
52	Instantaneous speed
54	Strides
55	Reserved
56	Reserved
58	CRC
59	ETX

5.4 Environmental Monitoring

Libelium [27] manufacturers and sells a ZigBee based development kit called Waspmote, shown in Figure 5.8. Waspmotes have a wide range of different sensors, and was therefore selected for implementation. Waspmotes consist of the basic platform board which has sensors for acceleration, battery rate, and temperature. These basic

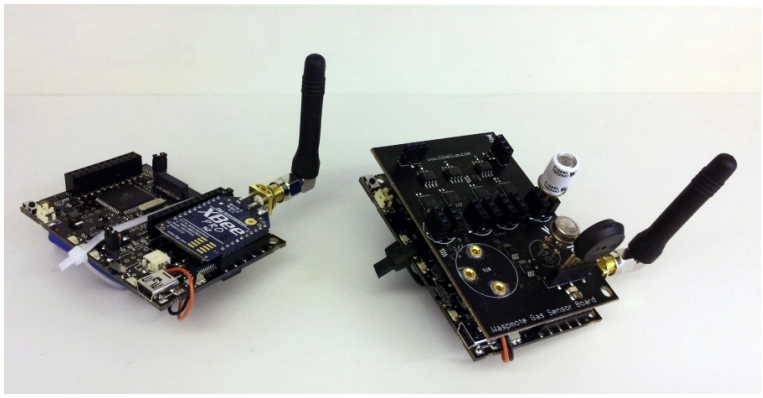

Figure 5.8: Waspmote nodes. Gas measurement extension board attached on the right one.

Waspmotes can be extended with different sensor boards, varying from ones measuring different gases, pressure and humidity to parking detectors and soil moistures. The development kit acquired contained sensor boards for gases, parking monitoring, electrical current measurements and a general input/output (IO) extension board. Different antennas and radio modules can also be connected for other technologies in addition to ZigBee and to vary the range of communication. With ZigBee radio the maximum range according to the manufacturer is over 10 kilometers. To read the measurements from the network a USB serial connection from Libelium was used. The USB device works as the ZigBee coordinator in the network.

All core functions, radio communications and sensor boards have well defined APIs for quick and easy deployment. The nodes had no reference software to send the measurements. Simple prototype node software was created for the nodes that reads the measurement from the sensors, forms a link to the coordinator and sends the

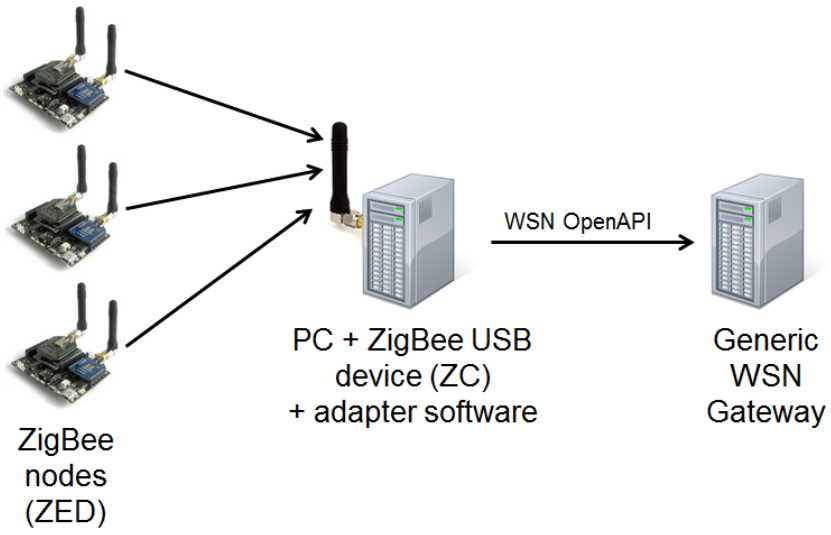

Figure 5.9: ZigBee network setup.

measurements with given intervals. The prototype software does not have support for multihop between the nodes. The support can be added later. An adapter software was written to handle the USB device, and communicate with the GWG. Adding multihop support for the nodes does not affect the adapter software. The adapter software was run on an small embedded PC. This setup is illustrated in Figure 5.9.

The first phase of integration was to create software for the nodes. The nodes contained different sensors but the basic functions were the same for all nodes. A simple software body was created for the basic functions and small additions for different sensors. The software on Waspmotes is divided into two functions: setup and loop. The setup function is run only once when the board is turned on and loop is run after that repeatedly. The phases of both functions and the flow of the software is illustrated in Figure 5.10. The setup function is used for initializing the sensors, real time clock and radio. The loop function reads the measurements, packages them in a ZigBee packet, sends them and sleeps for a given time. After sleep the loop starts again from the start.

Libelium provides an API to handle all ZigBee network functionality. Functions exist for initializing the radio, scanning for nodes, and sending and receiving packets. Listing 5.1 displays the code used for initializing the ZigBee radio. The API also

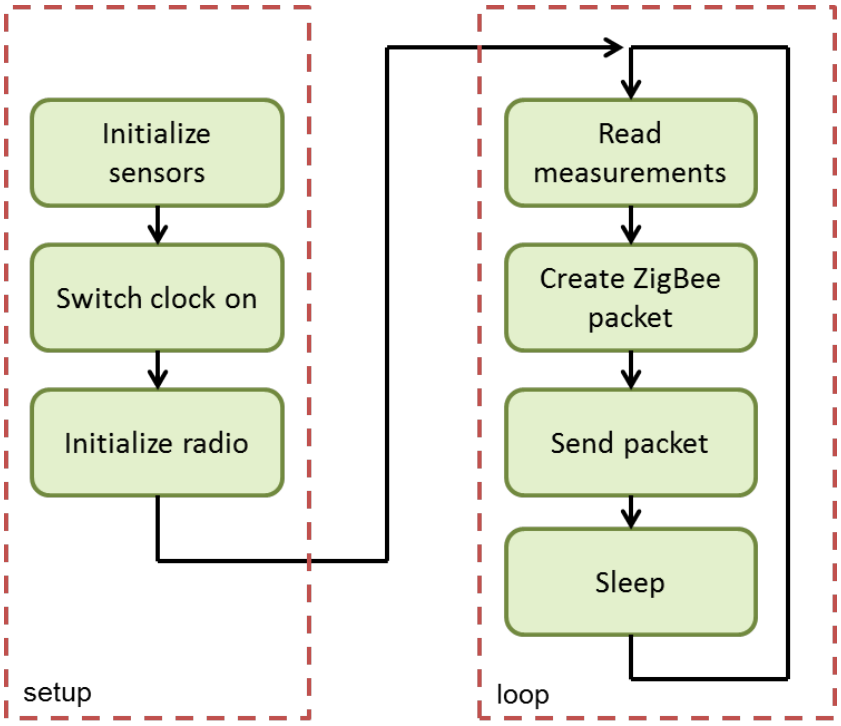

Figure 5.10: Flow of the Waspmote node software

provides functions for creating a ZigBee package and inserting the data to it. Listing 5.2 displays code for creating a ZigBee package and sending it to the coordinator.

```
1 // Inits the ZigBee library for right radio module
2 xbeeZB.init(ZIGBEE,FREQ2_4G,PRO);
3
4 // Set power level 0-4
5 xbeeZB.setPowerLevel(4);
6
7 // Power boost mode
8 // Increases receive by 1dBm, transmit by 2dBm
9 xbeeZB.setPowerMode(1);
10
11 // Powers ZigBee
12 xbeeZB.ON();
```

Listing 5.1: Waspmote code for initializing the ZigBee radio

```
1 // Creates ZigBee package
2 package=(packetXBee*) calloc(1,sizeof(packetXBee));
3
4 // Sets the source and destination information
5 xbeeZB.setOriginParams(package, NODE_ID, MY_TYPE);
6 xbeeZB.setDestinationParams(package, GATEWAY_MAC, (char*)data,
      MAC_TYPE,DATA_ABSOLUTE);
7
8 // Sends the package
9 xbeeZB.sendXBee(package);
```

Listing 5.2: Waspmote code for creating and sending a package

The development environment uses simple text strings to store data in packets. The API provides functions to add measurements in strings to the data area of the ZigBee packet. This method is not efficient since every character takes up 1 B. For example one measurement with a value of 1234 would take up 4 B or 32 b. If the value is presented in an integer it takes only 10 b. Therefore, measurement values are manually inserted as integers to the ZigBee package. The API uses American Standard Code for Information Interchange (ASCII) characters internally so a measurement value of 0 is interpreted as end of data.

A simple data packet format was created for use with the nodes, as presented in Table 5.5. The packet only has a timestamp and a pairs of measurement quantity and value for all measurements. The number of these value pairs varies depending on the number of sensors on the node. These packets are inserted into the data field of the

ZigBee package. The API uses American Standard Code for Information Interchange (ASCII) characters internally, so certain values have to be escaped before the data is inserted. For example a measurement value of 0 is interpreted as End of Data in ASCII.

Table 5.5: Example simple packet format.

Meaning	Timestamp	Quantity	Value	Quantity	Value
Value (hex)	0x0C051E090519	0x01	0x1234	0x02	0x0236
Value (decimal)	12-05-30 09:05:25	1	4660	2	566
Size	6B	1B	2B	1B	2B

In addition to the node software, the actual adapter software is needed. The adapter software was written in Java. It initializes the USB ZigBee coordinator and reads the measurements from the network. The basic Java JDK does not have native support for reading serial devices so an external library Java Communications API[49] was used. The WSN OpenAPI library was also used in this adapter to handle the GWG side of the adapter.

5.5 Home Automation

Z-Wave was selected as the home automation technology to integrate. Aeon Labs Z-Stick Series 2 [50] was selected as the gateway device because of the plug and play support in Linux and openzwave [29] support. A PC runs the adapter software and controls the USB gateway device. Z-Stick has a USB to serial converter and is therefore displayed as a simple serial port device on the PC. Controllable power switches, energy meter, flood detector, motion detector, door/window detector and smoke alarm devices were acquired for integration.

Openzwave library is provided with a sample software, called MinOZW, that sets up the Z-Wave network and receives measurements from it. Nothing is done to the measurements on the sample software. MinOZW is written in C and is used as a base for the Z-Wave adapter software, as it already handles the WSN side of the adapter. Implementation for data processing and GWG side of the adapter was created. The adapter has two threads, as illustrated in Figure 5.11. The Z-Wave thread initializes the Z-Wave network and then starts to listen for measurement messages. The Z-Wave thread acts as the WSN side of the adapter. When a message is received it is converted into WSN OpenAPI and sent to the GWG, if there is a connection. The API provides device IDs, which are mapped straight to logical nodeIDs, quantities and units in the messages so the conversion is straightforward. Messages are timestamped on arrival at the adapter software. The GWG thread

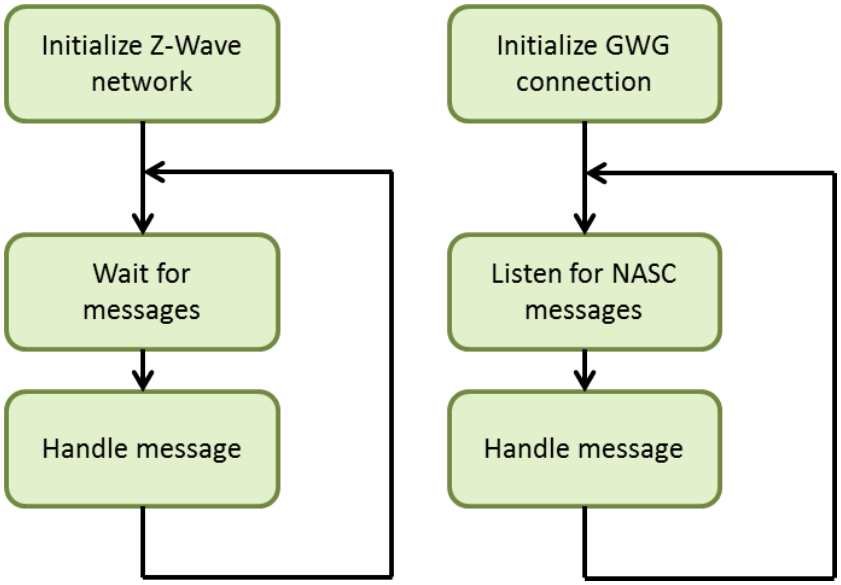

Figure 5.11: Flow of the Z-Wave adapter software

handles the the GWG side of the adapter. The WSN OpenAPI Java library could not be used because the adapter was implemented with C++, so the GWG side functionality was written from scratch. When a GWG connection is formed, the adapter listens for NASC messages.

The power switch nodes used in this thesis also contains actuators, they can be turned on and off. When the adapter receives a NASC message, it parsed and stored in an internal data model. Then the openzwave library is used to check if such node and actuator exist in the network. If they exists, a command is sent to the right device. This process is display in listing 5.3.

```
1 // Executes a received NASC messages from GWG.
2 void executeNASCMessage(NASCMessage* nasc)
3 {
4   // The nodeID in the NASC message
5   int nodeId_int = atoi((nasc->getNode()).c_str());
6
7   // Go trough the devices to find the one in NASC
8   for( list<NodeInfo*>::iterator it = g_nodes.begin(); it !=
        g_nodes.end(); ++it )
9   {
10    // Openzwave data model for storing node information
11    NodeInfo* nodeInfo = *it;
12
13    // When the wanted node is found...
14    if(( nodeInfo->m_homeId == g_homeId ) && ( nodeInfo->m_nodeId
         == nodeId_int ))
15    {
16      // ... go trough it's actuators to find the given one
17      for( list<ValueID>::iterator iter =
           nodeInfo->m_values.begin(); iter !=
           nodeInfo->m_values.end(); ++iter )
18      {
19        // If the right actuator is found, send the given command
             to it
20        if( Manager::Get()->GetValueLabel(*iter) ==
             nasc->getActuator() )
21        {
22          Manager::Get()->SetValue(*iter,nasc->getValue());
23        }
24      }
25    }
26  }
27 }
```

Listing 5.3: Z-Wave actuator control with openzwave library

Openzwave is still under development and contains bugs. Some nodes are not fully supported. The motion detector was recognized but did not send messages on events. The smoke detector needed a manual waking up by button pressing to start sending messages. Basic functionality of other nodes was right. The nodes support changing of some parameters, for example message sending interval. These parameters seemed to worked randomly. The problematic nodes were tested on an official commercial Z-Wave gateway device to rule out the possibility of broken devices.

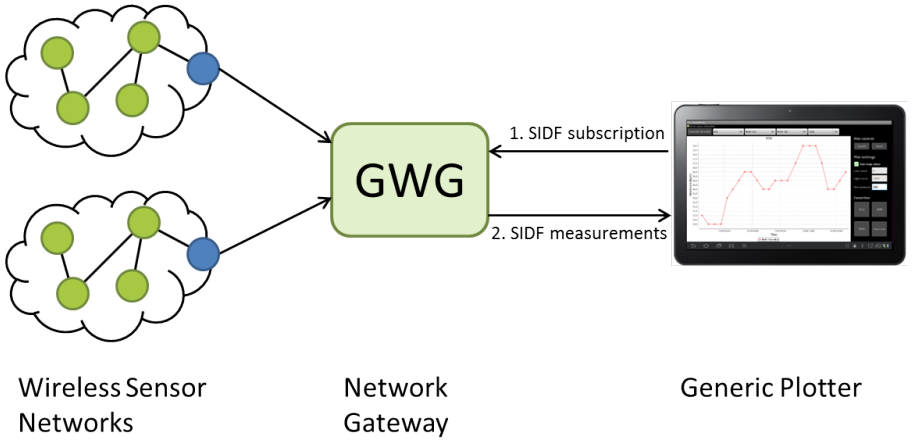

Figure 5.12: Generic plotter requesting measurement from the GWG

5.6 Technology Independent End-user Services

Technology independent end-user services were used to test the adapters. They were used to verify that all measurement data from different WSN technologies can be used similarly. Controlling of nodes should also be possible without knowing actual technologies or the physical node details.

Generic plotters are applications that request measurements from the GWG and interpret them graphically. These applications do not know from what technology certain measurements come from. The plotter sends a SIDF request to the GWG. The request can be for all measurements or just measurements from certain network or node. GWG receives the subscription and starts sending the wanted measurement to the plotter. The plotter then reads the data from the WSN OpenAPI messages and draws line or bar graphs. This process is illustrated in Figure 5.12.

5.6.1 Generic PC Plotter

A generic plotter for PC, called WSN Monitor, has been created earlier in the WAS project. WSN Monitor is written in Java and therefore runs on Windows and Linux. It was originally written as a proof of concept client to the GWG to demonstrate the WSN OpenAPI in use [40]. It has several different views, implemented as plug-ins to show the data in different ways. The software can show all the received measurements in a list, save them in a database and plot them in real time. All of the different views can be used at the same time and several graphs can be drawn also. The real time graph view of WSN Monitor is shown in Figure 5.13. The real time graph view uses an open source library JFreeChart for drawing the charts.

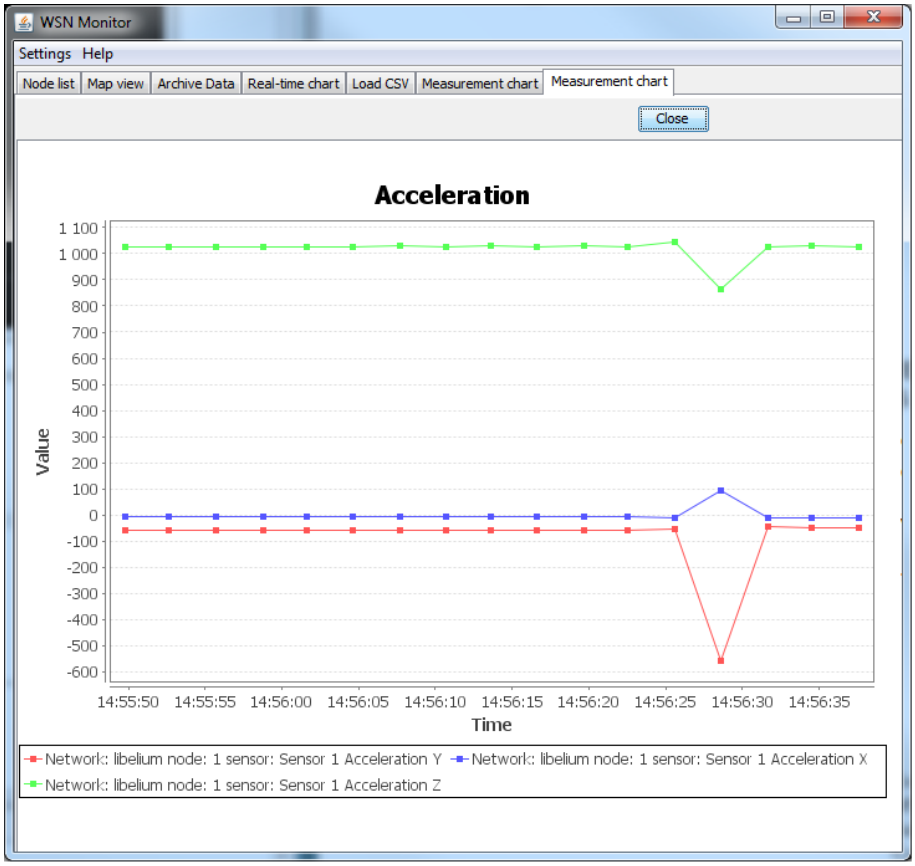

Figure 5.13: WSN Monitor real time graph view

5.6.2 Generic Android Plotter

WSN Monitor can only be run on PCs so a mobile application for drawing graphs was implemented for use with TUT MedSensor. Android was used as the platform again to so that already written MedSensor code could be reused. A generic plotter called TUT RemotePlot, shown in Figure 5.14, was written for Android tablets.

TUT MedSensor has only one view and draws only line graphs. An Android port of the JFreeChart library called AFreeChart was used to handle the graph drawing. Bar graphs could be added to the application with reasonable work. RemotePlot sorts the received messages by node, sensor, quantity and component in the menus at the top. The menus fill with content as it arrives to the application. These menus can be used to select wanted measurements to be drawn. The application also contains four quick selection buttons that can be preconfigured to request and draw only wanted measurement. This is to save bandwidth and amount of data to handle since mobile devices have limited amount of processing power.

During the usage of the application it was noticed that drawing measurements with high sample rate, for example the ECG with 50Hz, makes the application very unre-

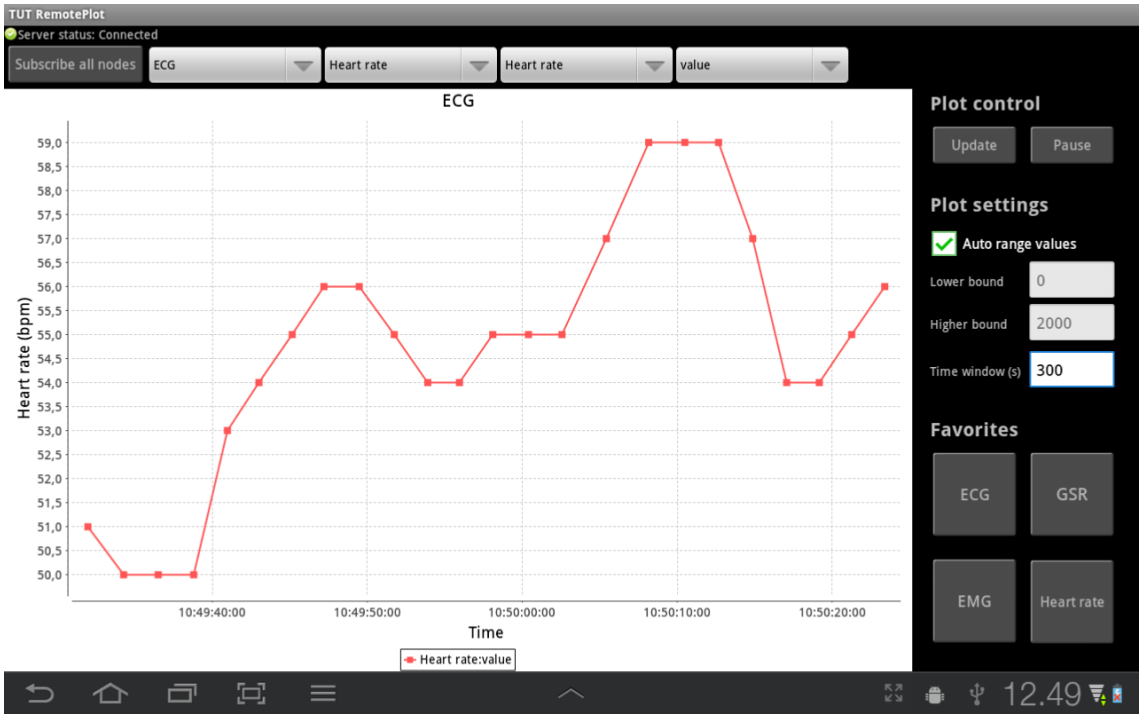

Figure 5.14: TUT RemotePlot Android generic plotter

sponsive and slow. Processing time usage of different components of the application were measured to see what is the bottleneck of the application. Depending on the data sample rate and amount of components drawn at the same time, the graph drawing took 80-99% of the processing time. AFreeChart is originally meant for displaying only static data and not for constantly updating dynamic data. There are only few ready made libraries available for graph drawing on Android and most of them with inadequate performance or functionalities. To make the application usable with high sample rates a graph library would have to be created from scratch or one of the open source libraries optimized.

6. RESULTS AND EVALUATION

The adapter complexity is displayed in Table 6.1 with the total amount of lines of code and runtime memory usage. As can be seen, Z-Wave adapter takes both more lines of code and memory. This is due to the Openzwave library which supports a wide range of different nodes and gateway devices. The Bluetooth and ZigBee adapters can be run on low performance embedded devices.

Table 6.1: Lines of code and runtime memory usage of the adapters.

Adapter	Lines of code	Memory usage (MB)
Bluetooth	3026	27
ZigBee	2052	31
Z-Wave	50178	48

Measurement packet sizes in bytes are displayed in Table 6.2. The table shows sizes of the packets in technology specific native format and in WSN OpenAPI. The WSN OpenAPI takes more space than the native formats because the measurements are stored as text in XML, as with the native formats values are stored as integers. The WSN OpenAPI messages also contain more information, such as timestamps, and measurement quantities and sensor names as text. The messages are usually transported from the adapter to the GWG over wide bandwidth connection, such as WLAN, making the effect of increased packet size small. WSN OpenAPI packet sizes could be decreased by using the CSV format.

Table 6.2: Native and WSN OpenAPI measurement packet sizes.

Adapter	Native format (B)	WSN OpenAPI (B)
Bluetooth (Shimmer, accel+ECG)	12	593
Bluetooth (Zephyr)	60	309
ZigBee (acceleration+temperature)	23	547
Z-Wave	15	309

7. CONCLUSIONS

This thesis presents the design, implementation and evaluation of WSN gateway adapters. Adapters separate the gateway integration process of new WSN technologies from the development of the actual gateway and end-user services. Three prototype adapters were created for an existing end-to-end WSN architecture.

A general model for an gateway adapter was designed. The model unifies the integration process for all new technologies by dividing the adapter into WSN technology specific side and gateway side. It enables the reuse of software components, especially on the gateway side. A generic WSN integration process and it's phases are explained but more detailed technology specific instructions can not be given, because of the wide range of technologies and devices. The model and the integration process can be used for future adapter development.

Implemented adapters were successfully evaluated with generic data plotters. All adapters worked throughout the test period and did not interfere with each other. Adapters were developed concurrently with the end-user services.

The biggest challenges in adapter development are in the technology specific parts of the adapter. Some WSNs are poorly documented, especially the packet formats and timestamping of measurements. Acquiring all required data and converting it into WSN OpenAPI takes most of the time in the implementation. Some WSN technologies are also closed source and require the use of third party components. This brings more unreliability factors to the implementation and also to the usage of the adapter.

The objective of the thesis was to integrate several WSN technologies to the GWG and form a working end-to-end multi-technology WSN architecture. This objective was achieved using existing softwares together with the implemented adapters. More complex and intelligent adapters could have been implemented to achieve more automatic integration process.

As future work the adapters could do mapping from physical devices to logical nodes automatically by communicating with GWG and other adapters. All nodes would

be described in XML from which the GWG would gather information for mapping. This would make the deployment of complete end-to-end architecture easier and faster. The automatic mapping functionality can be added to existing adapters. It would be interesting to use adapters with other WSN gateway softwares. This would only require changing the gateway side components of the adapters to match the new gateway. More technologies will also be integrated in the future. The adapters will be used in a pilot home automation application.

REFERENCES

[1] R. Beckwith, D. Teibel, and P. Bowen. Report from the field: results from an agricultural wireless sensor network. In Local Computer Networks, 2004. 29th Annual IEEE International Conference on, page 471-478, Nov 2004.

[2] R. Hongliang, M.Q.-H. Meng and C. Xijun, Physiological information acquisition through wireless biomedical sensor networks, Information Acquisition, IEEE International Conference on, Jul 2005

[3] M.A. Hussain, WSN Research Activities for Military Application, Advanced Communication Technology, 11th International Conference on, Vol 1, page 271-274, Feb 2009

[4] Elisa, Oyj. Elisa Vahti service. Available: `http://www.elisa.fi/vahti/`, Referenced on April 1st 2012.

[5] I.F. Akyildiz, Weilian Su, Y. Sankarasubramaniam and C. Erdal, A Survey on Sensor Networks, Communications Magazine, IEEE, 40(8):102-114, Aug 2002.

[6] M. Kuorilehto, M. Kohvakka, J. Suhonen, P. Hämäläinen, M. Hännikäinen, and T. D. Hämäläinen. Ultra-Low Energy Wireless Sensor Networks in Practice. John Wiley & Sons Ltd, Chichester, 2007.

[7] Jukka Suhonen, Mikko Kohvakka, Marko Hännikäinen, and Timo D. Hämäläinen. Design, Implementation, and Experiments on Outdoor Deployment of Wireless Sensor Network for Environmental Monitoring. In Proc. Embedded Computer Systems: Architectures, Modeling, and Simulation, Samos, Greece, July 2006, page 109-121, New York, NY, USA, 2006. Springer Berlin / Heidelberg.

[8] M. Kuorilehto, System Level Design Issues in Low-Power Wireless Sensor Networks. PhD thesis, Tampere University of Technology, Tampere, 2008

[9] T. Wark, C. Crossman, et. al, The Design and Evaluation of a Mobile Sensor/Actuator Network for Autonomous Animal Control, Information Processing in Sensor Networks, pages 206 -215, April 2007

[10] Vivonoetics Equivital, Available: `http://vivonoetics.com/products/sensors/equivital/`, referenced on April 1st 2012

[11] F. Amato, V. Casola, A. Caglione and A. Mazzeo, A Common Data Model for Sensor Network Integration, Complex, Intelligent and Software Intensive Systems (CISIS), 2010 International Conference on, page 1081-1086, Feb 2010

[12] ZigBee Alliance, Available: http://zigbee.org/, referenced on April 30th 2012

[13] Z-Wave, Available: http://www.z-wave.com, referenced on August 16th 2012

[14] Wavenis, Available: http://www.coronis.com/en/wavenis_technology.html, referenced on August 31st 2012

[15] EnOcean, Available: http://www.enocean.com, referenced on August 31st 2012

[16] Bluetooth, Available: http://www.bluetooth.com, referenced on August 16th 2012

[17] ANT, Available: http://www.thisisant.com/, referenced on August 31st 2012

[18] WirelessHART, Available: http://www.hartcomm.org/, referenced on August 31st 2012

[19] ISA100, Available: http://www.isa.org/isa100, referenced on August 31st 2012

[20] DASH7, Available: http://www.dash7.org, referenced on August 31st 2012

[21] RuBee, Available: http://www.rubee.com, referenced on August 31st 2012

[22] WTRS Wireless Sensor Network Technology Trends Q1 2012, Feb 2012, 257 pages, Available: http://www.researchandmarkets.com/reportinfo.asp?report_id=2090271

[23] T. Sato, A scatternet operation protocol for Bluetooth ad hoc networks, The 5th International Symposium on Wireless Personal Multimedia Communications, pages 223-227, Vol 1, Oct 2002

[24] Bluetooth 2.1 Specification, Available: https://www.bluetooth.org/Technical/Specifications/adopted.htm, referenced on August 18 2012

[25] IEEE 802.15.4d standard, Available: http://standards.ieee.org/getieee802/download/802.15.4d-2009.pdf

[26] ZigBee Specification, Available: http://www.zigbee.org/Specifications/ZigBee/download.aspx, referenced on August 18 2012

[27] Libelium, Avalaible: http://www.libelium.com, referenced on May 23rd 2012

[28] Sigma Designs, Available: http://www.sigmadesigns.com/, referenced on May 29th 2012

[29] open-zwave project, Available: http://code.google.com/p/open-zwave/, referenced on May 29th 2012

[30] Z-Wave Insiders Wiki, Available: http://wiki.zwaveeurope.com/, reference on August 18th 2012

[31] Everspring ST814 temperature/humidity detector datasheet, Available: http://www.homeseer.com/pdfs/Everspring/ST814.pdf, referenced on August 31st 2012

[32] Everspring ST812 flood detector datasheet, Available: http://www.homeseer.com/pdfs/Everspring/ST812.pdf, referenced on August 31st 2012

[33] Universal Remote Control, Inc., Available: http://www.universalremote.com/products/residential/remotes/mx-880, referenced on May 23rd 2012

[34] Home Automation, Inc., Available: http://www.homeauto.com/Products/Omnistat/Omnistat2Products.asp, referenced on May 23rd 2012

[35] V. Casola, A. Gaglione, A. Mazzeo, A Reference Architecture for Sensor Networks Integration and Management, GSN '09 Proceedings of the 3rd International Conference on GeoSensor Networks, pages 158-168, 2009

[36] Ahn, S., Chong, K.: Building a Bridge for Heterogeneous Sensor Networks. In: Proceedings of the Fourth IEEE Workshop on SEUS-WCCIA 2006

[37] Aberer, K., Hauswirth, M., Salehi, A.: The Global Sensor Networks middleware for efficient and flexible deployment and interconnection of sensor networks, Technical Report, 2006

[38] Gibbons, P.B., Karp, B., Ke, Y., Nath, S., Seshan, S.: IrisNet: An Architecture for a World- Wide Sensor Web. IEEE Pervasive Computing 2(4), 2003

[39] Jeff Shneidman, Peter Pietzuch, Jonathan Ledlie, Mema Roussopoulos, Margo Seltzer, and Matt Welsh. Hourglass: An infrastructure for connecting sensor networks and applications. Technical report, Harvard University, 2004. Harvard Technical Report TR-21-04.

[40] Olli Kivelä, Open Interfaces for Wireless Sensor Networks, Master of Science Thesis, TUT, 2012

[41] Eclipse, Available: http://www.eclipse.org, referenced on August 16th 2012

[42] MiKTeX, Available: http://www.miktex.org, referenced on August 16th 2012

[43] Android, Available: http://www.android.com, referenced on August 16th 2012

[44] Google's Android becomes the world's leading smart phone platform, Canalys. Jany 31 2011. Available: http://www.canalys.com/newsroom/google%E2%80%99s-android-becomes-world%E2%80%99s-leading-smart-phone-platform

[45] Android Developers, Available: http://developer.android.com, referenced on August 16th 2012

[46] Shimmer Research, Available: http://www.shimmer-research.com/, referenced on August 18th 2012

[47] TinyOS, Available: http://www.tinyos.net/, referenced on August 31st 2012

[48] Zephyr HxM, Available: http://www.zephyr-technology.com/consumer-hxm, referenced on May 29th 2012

[49] Java Communications API, Available: http://www.oracle.com/technetwork/java/index-jsp-141752.html, referenced on June 6th 2012

[50] Aeon Labs Z-Stick Series 2, Available: http://www.aeon-labs.com/site/products/view/2/, referenced on August 18th 2012